EDIT GRASSIANI

PROCEDIMENTOS MICROPROGRAMADOS NUM MINICOMPUTADOR

"Dissertação de Mestrado" apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de MESTRE EM ENGENHARIA.

Área de concentração: Engenharia de Eletricidade

Orientador: Prof. Dr. Antônio Hélio Guerra Vieira

São Paulo

1975

FD-104

EDIT GRASSIANI

PROCEDIMENTOS MICROPROGRAMADOS NUM MINICOMPUTADOR

"Dissertação de Mestrado" apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de MESTRE EM ENGENHARIA.

Área de concentração: Engenharia de Eletricidade

Orientador: Prof. Dr. Antônio Hélio Guerra Vieira

BIBLIOTEC
Dept.º Eng. ELETRI IDADE
E. P. U. S. P

São Paulo

1975

A meus pais.

AGRADECIMENTOS

Ao Prof. Dr. Antonio Hélio Guerra Vieira pelo constante estímulo e orientação.

Ao Prof. Dr. Antonio Marcos de Aguirra Massola, Prof. Eng. Lucas Antonio Moscato e Eng. Geraldo Lino de Campos pelo apoio.

À Fundação para o Desenvolvimento Tecnológico da Engenharia, que colocou à disposição os recursos necessários à elaboração deste trabalho.

Aos Engs. Roger Carlos Cintra Ferreira, Stephan Kovach, João José Neto e Paulo Kioshi Endo pela colaboração.

À secretária Maria Inês Mello pela constante dedicação e esforço no serviço de datilografia.

À desenhista Rosa Terezinha Mendes pelo serviço de desenho.

À toda a equipe de engenheiros e estagiários que direta ou indiretamente cooperaram para a realização deste trabalho.

SINÓPSE

Após considerações gerais sobre microprogramação, esta dissertação apresenta as técnicas de microprogramação adotadas no computador G-10. São discutidos os compromissos entre implementação por hardware, software e microprogramação, bem como a otimização das instruções no G-10. Estuda-se a emulação do computador IBM 1130 concluindo-se por sua viabilidade, e analisa-se o desempenho resultante. Apêndices apresentam as micro-ordens e listagens de exemplos de microprogramas do G-10 e do emulador do IBM 1130.

ABSTRACT

After a brief introduction to microprogramming,this thesis describes some microprogramming techniques used in the G-10 computer project. Hardware, software and microprogram im plementation trade-offs are examined, as well as instruction otimization in G-10. The emulation of the IBM 1130 computer is studied, with conclusions on its feasibility. The performance of the resulting system is analyzed. Appendices show the micro--orders and some listings of G-10 and emulator microprograms.

ÍNDICE

I INTRODUÇÃO

II CONSIDERAÇÕES GERAIS SOBRE MICROPROGRAMAÇÃO

II.1 HISTÓRICO
II.2 DEFINIÇÕES
II.3 TIPOS DE MICROPROGRAMAÇÃO

III EXEMPLO DE UMA UNIDADE DE CONTROLE MICROPROGRAMADA

III.1 CARACTERÍSTICAS FUNCIONAIS DO G-10
III.2 FLUXO DE INFORMAÇÕES NA UNIDADE DE CONTROLE

IV MICROPROGRAMAÇÃO DAS INSTRUÇÕES

IV.1 FASES DA MICROPROGRAMAÇÃO DE UMA INSTRUÇÃO
IV.2 TIPOS DE MICROPROGRAMAS
IV.3 DEPURAÇÃO DOS MICROPROGRAMAS

V OTIMIZAÇÃO DA MICROPROGRAMAÇÃO

V.1 OTIMIZAÇÃO EM TEMPOS DE ESPAÇO
V.2 OTIMIZAÇÃO EM TERMOS DE TEMPO

VI CARACTERÍSTICAS IMPLEMENTADAS EM SOFTWARE/HARDWARE/ MICROPROGRAMAÇÃO - COMPROMISSO E RESTRIÇÕES

VII EXEMPLO DE APLICAÇÃO DE MICROPROGRAMAÇÃO: EMULAÇÃO

VII.1 INTRODUÇÃO
VII.2 ESTUDO COMPARATIVO DAS VÁRIAS TÉCNICAS PARA A IMPLANTAÇÃO DE PROGRAMAS DE UMA MÁQUINA EM OUTRA
VII.3 DEFINIÇÕES FUNCIONAIS E PROCEDIMENTAIS
VII.4 MÁQUINA HOSPEDEIRA E MÁQUINA HÓSPEDE
VII.5 TÉCNICAS DE EMULAÇÃO

Í N D I C E
(conclusão)

VIII EXEMPLO DE EMULAÇÃO - SISTEMA IBM 1130

VIII.1 CARACTERÍSTICAS PRINCIPAIS DO IBM 1130/2D
VIII.2 EMULAÇÃO DAS INSTRUÇÕES DO IBM 1130
VIII.3 TEMPOS DE EMULAÇÃO
VIII.4 CRITÉRIOS DE AVALIAÇÃO DO EMULADOR
VIII.5 EMULAÇÃO DE INSTRUÇÕES DE ENTRADA E SAÍDA

IX CONCLUSÕES

APÊNDICE A

MICRO-ORDENS DO G-10

APÊNDICE B

EXEMPLOS DE MICROPROGRAMAS DO G-10

APÊNDICE C

EXEMPLOS DE MICROPROGRAMAS DO EMULADOR DO IBM 1130

I - INTRODUÇÃO

O objetivo deste trabalho é apresentar algumas técnicas, procedimentos e funções implementáveis num minicomputador com uma unidade de controle microprogramada.

A microprogramação é uma técnica recentemente desenvolvida [3, 5, 13, 14], utilizada nos mais modernos minicomputadores que atualmente estão sendo utilizados para as mais diversas aplicações nos ramos científicos, comerciais e industriais.

Uma das grandes vantagens de um processador microprogramado é a flexibilidade que o sistema adquire no sentido de facilitar a sua adaptação à aplicações específicas, tais como controle de processos, processamento em tempo real e tempo compartilhado, aquisição de dados, monitoramento de ensaios de laboratório, estudos didáticos em universidades, etc., com quase exclusivamente a mudança de microprogramas na memória de controle.

No Capítulo II, são apresentados um breve histórico da microprogramação e as principais definições de termos empregados ao longo do trabalho.

No Capítulo III, são descritas as características da unidade de controle de um minicomputador microprogramado desenvolvido no Laboratório de Sistemas Digitais da Escola Politécnica da Universidade de São Paulo, o Sistema G-10, que serviu de referência para este trabalho.

No Capítulo IV, discute-se sobre a técnica de implementação dos microprogramas correspondentes ao conjunto de instruções do G-10, sua modularidade e algorítmos em geral.

No Capítulo V, são apresentadas algumas técnicas utilizadas na otimização do conjunto de microprogramas quanto ao número de palavras de controle, e quanto ao tempo de execução.

O Capítulo VI consta de um breve estudo sobre a escolha e atribuição de funções a cada aspecto de implementação de um mini

computador: Hardware, Software e Microprogramação. São exemplificadas algumas funções atribuídas à microprogramação e implementadas no G-10.

O Capítulo VII apresenta um exemplo de aplicação da microprogramação, a Emulação, constando de um estudo comparativo de várias técnicas de processar um programa de um computador (hóspede) em outro (hospedeiro). São apresentadas as definições de emulação, e um estudo sobre o grau de compatibilidade necessário entre os sistemas hóspede e hospedeiro.

No Capítulo VIII há um exemplo prático de implementação de um emulador de instruções do sistema IBM 1130 no G-10.

O Capítulo IX apresenta as conclusões a que se chegou durante o transcorrer do trabalho, e sugestões para pesquisas futuras no campo da microprogramação.

O Apêndice A apresenta as micro-ordens do G-10.

O Apêndice B consta de listagens de microprogramas implementados no G-10.

O Apêndice C contém microprogramas do emulador do IBM 1130.

II - CONSIDERAÇÕES GERAIS SOBRE MICROPROGRAMAÇÃO

II.1 HISTÓRICO

De uma maneira geral, um computador digital se subdivide em quatro partes funcionais, a saber:

- unidade lógica e aritmética;
- unidade de armazenamento (memória);
- unidade de controle;
- unidade de entrada e saída.

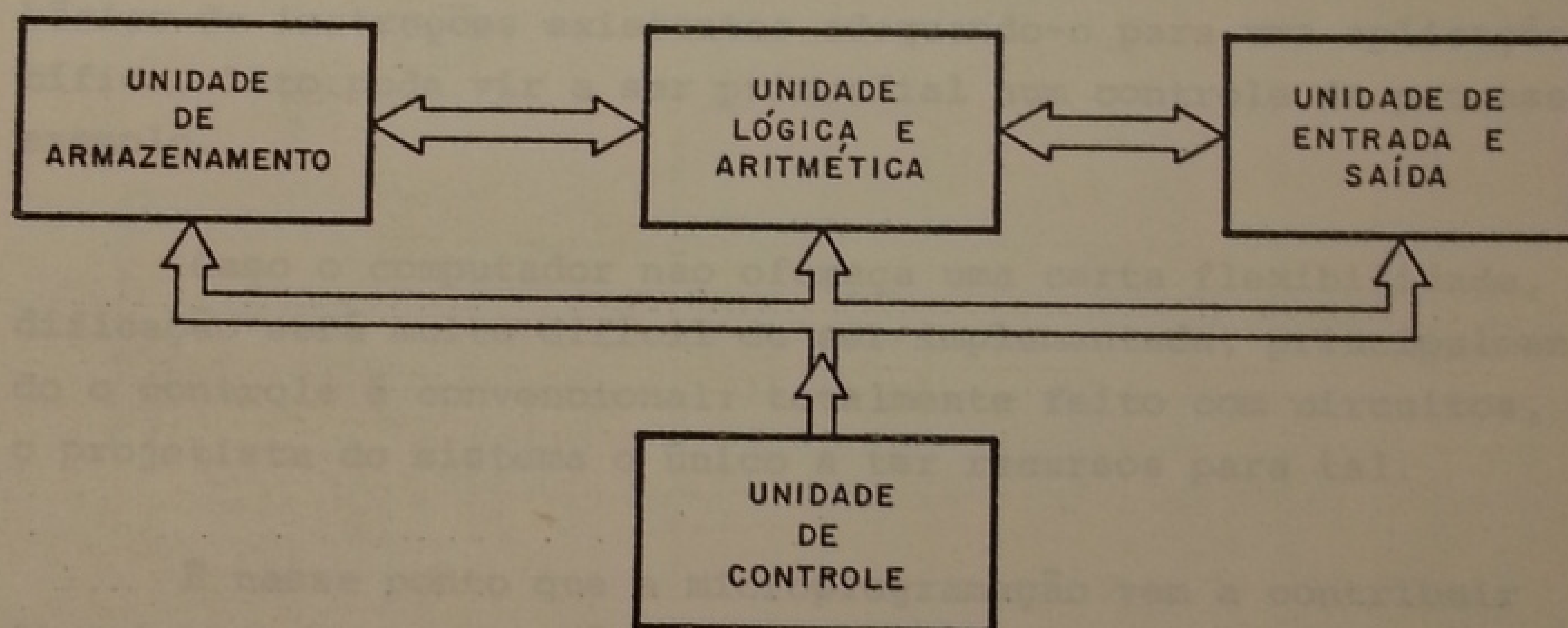


Figura II.1 - Blocos Funcionais de um Computador.

A unidade de controle é responsável pelo gerenciamento da operação interna do computador e tem as seguintes funções:

- busca da instrução de máquina;
- decodificação e interpretação da instrução;

- ativação dos elementos lógicos para permitir o fluxo de informações pelas unidades necessárias para a execução da instrução;
- finalização da execução da instrução, atendimento à interrupções e ativação dos circuitos para a fase de busca de uma nova instrução.

A unidade de controle, implementada convencionalmente segundo as finalidades acima citadas, consiste de uma série de elementos: portas ("gates"), registradores, "flip-flops", multiplexadores, etc., interligados entre si de uma maneira desordenada e não sistemática. Esta ausência de modularidade não permite que alguma característica da máquina seja facilmente modificada quando da necessidade de acrescentar alguma instrução nova que possa vir a enriquecer o conjunto básico de instruções existentes adequando-o para uma aplicação específica. Isto pode vir a ser primordial num controle de processos, por exemplo.

Caso o computador não ofereça uma certa flexibilidade, a modificação será muito difícil de ser implementada, principalmente quando o controle é convencional: totalmente feito com circuitos, sendo o projetista do sistema o único a ter recursos para tal.

É nesse ponto que a microprogramação vem a contribuir significantemente na ordenação e sistematização da implementação de uma unidade de controle.

A concepção de um controle programado numa unidade de armazenamento não destrutivo se deve a M.V. Wilkes [5], da Cambridge University Mathematical Laboratory, no ano de 1950. O objetivo de Wilkes e seu grupo era o de reorganizar os componenetes de um computador numa ordem sistemática, de maneira a possibilitar maior modularidade e maior facilidade de manutenção para os circuitos de um computador.

Wilkes notou que cada instrução de máquina podia ser subdividida em uma série de operações elementares que efetuassem o contro

le sobre os vários elementos lógicos da máquina (portas, multiplexadores, registradores, etc.), responsáveis pela transferência de informações no processador. Estes elementos seriam controlados por sinais gerados periodicamente por uma unidade armazenadora de informações, que contivesse toda a seqüência lógica de operações sob forma programada, necessária à execução de determinadas atividades.

O esquema geral da idéia de Wilkes está ilustrado na figura II.2.

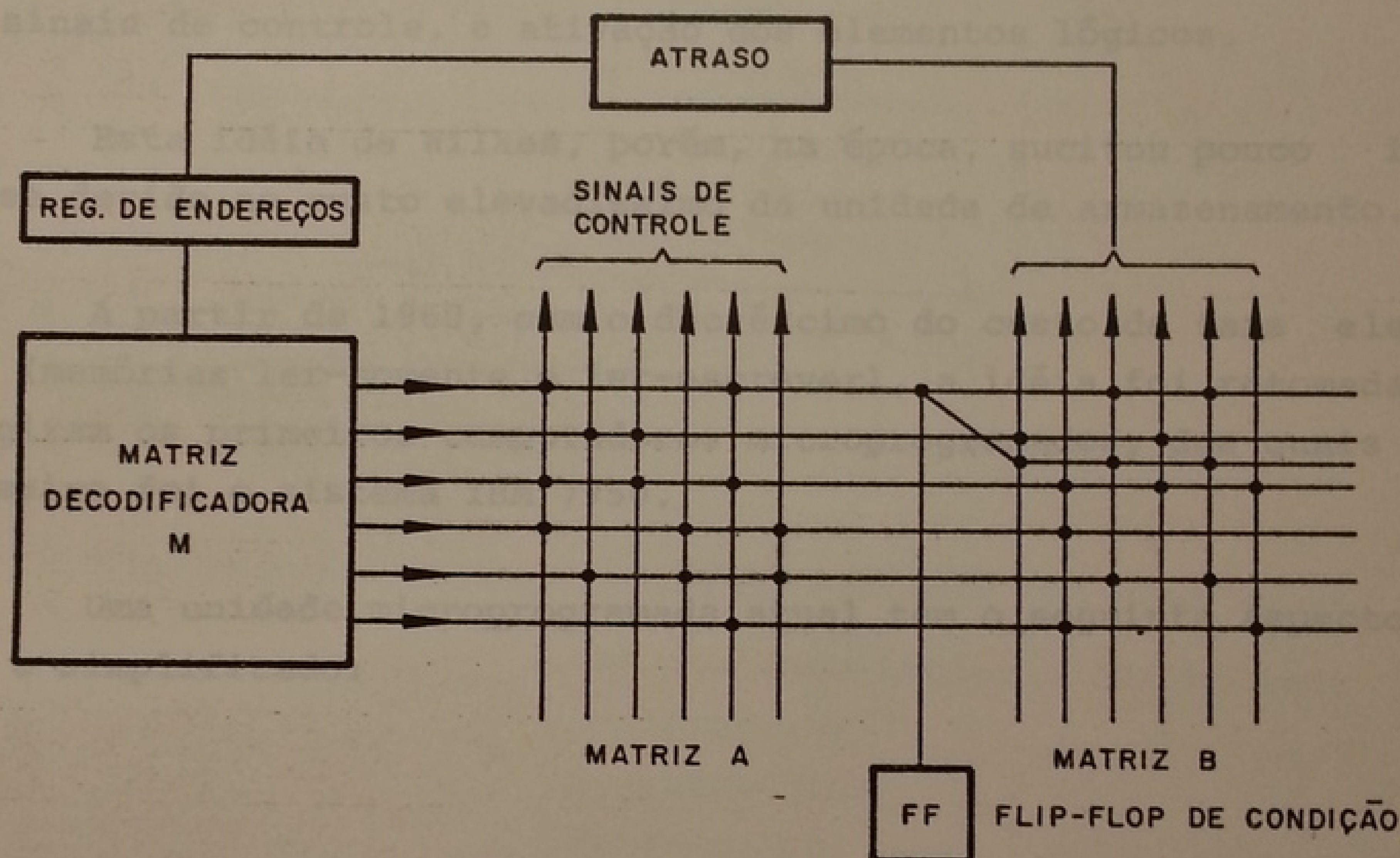


Figura II.2 - Esquema Geral da Idéia de Wilkes

A unidade armazenadora da seqüência lógica de operações a serem executadas é representada pela MATRIZ DECODIFICADORA. Esta matriz possui suas entradas ligadas a um REGISTRADOR DE ENDEREÇOS que seleciona uma das linhas da matriz. As saídas são conectadas a uma série de linhas condutoras (matrizes A e B), de tal maneira que a cada linha de saída da matriz decodificadora está associada a ativações (pontos de ligação) de determinados sinais que irão agir

sobre os elementos lógicos da máquina.

A matriz A gera os sinais de controle propriamente ditos.

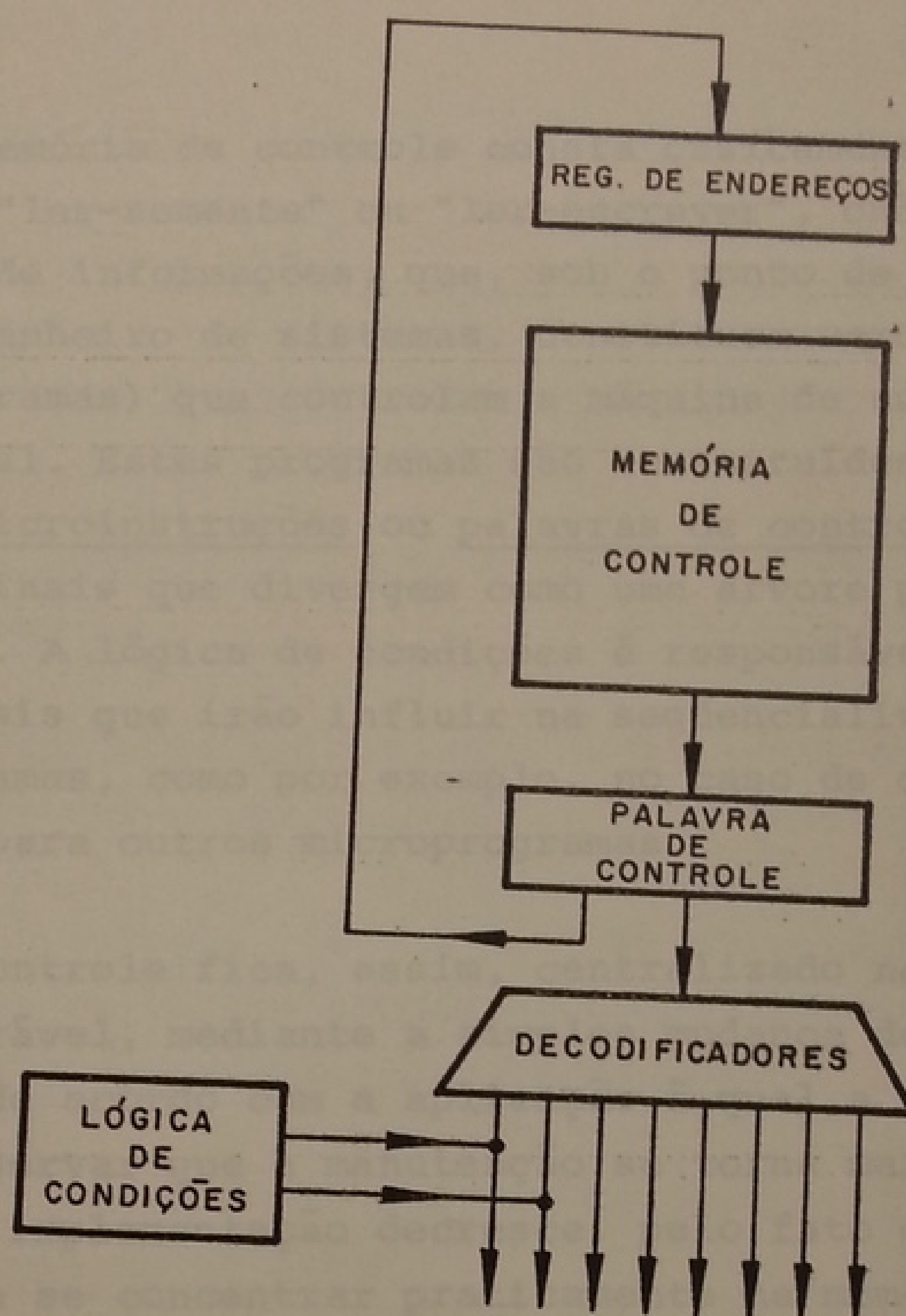
A matriz B gera sinais que especificam o próximo endereço a ser selecionado na matriz decodificadora. A geração desse endereço está condicionada a um teste representado num flip-flop. Se o teste for verdadeiro, um próximo endereço diferente é selecionado.

O atraso global do sistema é representado por um bloco de atraso, que determinaria a duração de um ciclo completo de geração de sinais de controle, e ativação dos elementos lógicos.

Esta idéia de Wilkes, porém, na época, sucitou pouco interesse devido ao custo elevadíssimo da unidade de armazenamento.

A partir de 1960, com o decréscimo do custo de tais elementos (memórias ler-somente e ler-escrever), a idéia foi retomada, e surgiram os primeiros computadores microprogramados, dos quais o primeiro foi o sistema IBM 7950.

Uma unidade microprogramada atual tem o seguinte aspecto geral e simplificado:



Sinais de controle que agem sobre:

ULA (Unidade Lógica e Aritmética)
Deslocadores
Registradores
Processador
Memória
Entrada e Saída
Própria Unidade de Controle

Figura II.3 - Unidade de Controle Microprogramada atual

A memória de controle consta basicamente de uma memória rápida tipo "ler-somente" ou "ler-escrever", onde estão armazenadas uma série de informações, que, sob o ponto de vista de um programador ou engenheiro de sistemas, constituem verdadeiros programas (microprogramas) que controlam a máquina de uma maneira ordenada e seqüencial. Estes programas são constituídos de uma série de instruções (microinstruções ou palavras de controle) que são decodificadas em sinais que divergem como uma árvore para as várias partes da máquina. A lógica de condições é responsável pela geração de certos sinais que irão influir na seqüencialização de execução dos microprogramas, como por exemplo, no caso de decisões e desvios condicionais para outros microprogramas.

O controle fica, assim, centralizado na máquina, e é facilmente alterável, mediante a simples mudança de um ou outro microprograma, de acordo com a aplicação à qual a máquina é destinada. Pode-se observar que a manutenção se torna mais fácil, assim como o custo de implementação decresce, pelo fato da lógica da unidade de controle se concentrar praticamente na memória de controle.

Com o advento da microprogramação surge uma nova atividade a do microprogramador. Este deve estar bem familiarizado com a estrutura interna da máquina, às vezes até ao nível de sinais, a fim de ordenar efetivamente a seqüencialização de operações nos microprogramas. Ao mesmo tempo, ele deve ter conhecimentos de programação normal de um computador, de maneira a poder escrever microprogramas corretos e eficientes. Atualmente, muitas funções que o software outrora exercia, são implementadas em microprogramas, tornado assim, estas funções automáticas e bem mais eficientes em termos de tempo de execução (vide Capítulo VI).

II.2 DEFINIÇÕES

A seguir são apresentadas as definições para certos termos que são largamente utilizados neste trabalho.

Memória de Controle - é a unidade de armazenamento que contém microprogramas.

Microprograma - é um conjunto de palavras de controle ou microinstruções que devem ser executadas numa determinada seqüência.

Palavra de Controle - é uma palavra da unidade de armazenamento, subdividida em vários campos contendo micro-ordens, que podem ser executados em um ou mais ciclos de UCP.

Microinstrução - é uma palavra de controle, cujas micro-ordens são executadas em um só ciclo.

Campo - é cada subdivisão da palavra de controle onde estão especificadas as micro-ordens.

Micro-ordem - é um código particular de uma certa função especificado num dos campos da palavra de controle.

Sinal de Controle - é um sinal lógico elementar que se origina da decodificação de cada micro-ordem. Estes sinais divergem da unidade de controle para partes diferentes da máquina, permitindo ou inibindo a passagem de informação pelos elementos armazenadores do fluxo de dados e demais circuitos, além da própria unidade de controle.

II.3 TIPOS DE MICROPROGRAMAÇÃO

A classificação dos diversos tipos de microprogramação depende, em geral, da maneira pela qual é implementada a palavra de controle numa máquina. O problema de definição da palavra de controle é semelhante ao problema da definição de um conjunto de instruções de uma máquina, quanto ao formato, campos e comprimento das instruções.

Primeiramente, o projetista-microprogramador precisa conhecer a fundo o hardware, o sincronismo entre as unidades funcionais, e preparar uma lista de todos os elementos que serão controlados pelos microprogramas, tais como controles sobre os registradores, deslocadores, funções da unidade aritmética, acessos à memória, aos canais de entrada e saída, registradores de status, etc.

Os elementos desta lista (sinais de controle) podem ser agrupados em conjuntos, de tal maneira que cada conjunto é responsável pela ativação de blocos lógicos num determinado local da máquina. Por exemplo, um dos conjuntos de sinais poderá exercer o controle sobre uma das entradas da unidade lógica aritmética (ULA), selecionando um registrador, cujo conteúdo será manipulado durante a operação. Outro conjunto selecionaria outro registrador cujo conteúdo será manipulado pela segunda entrada da ULA. Um terceiro conjunto especificaria a operação a ser efetuada pela ULA, e assim por diante.

Dentro de cada conjunto, os sinais devem ser mutuamente exclusivos, isto é, somente um dos sinais estaria ativo durante um determinado ciclo da UCP. Por exemplo, no conjunto que especifica as operações da ULA, não se pode especificar as operações de adição e subtração ao mesmo tempo, o que é irrealizável.

Esta é a técnica para a configuração de uma palavra de controle. Os conjuntos de sinais mutuamente exclusivos constituem os campos de uma palavra de controle, e os sinais de cada conjunto (ou campo), que podem ser codificados dentro de um mesmo campo são as micro-ordens.

Um exemplo simples de palavra de controle é a do HP2100 [5]

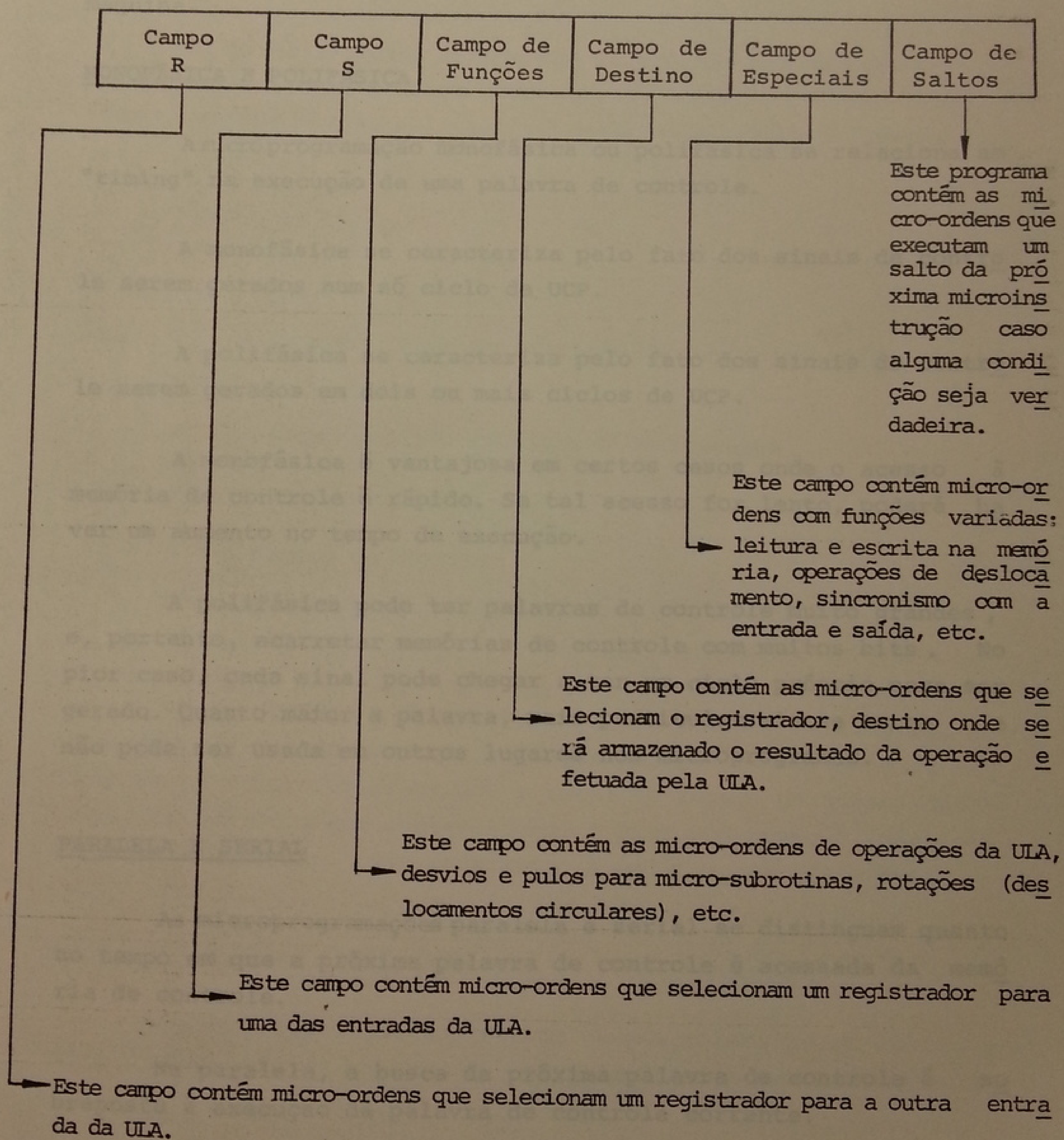


Figura II.4 - Palavra de Controle do HP2100

A seguir serão descritos os vários tipos de microprogramação com as vantagens e desvantagens de cada um, cabendo ao projetista a escolha do tipo que melhor se adapte às necessidades da máquina.

MONOFÁSICA E POLIFÁSICA

A microprogramação monofásica ou polifásica se relaciona ao "timing" na execução de uma palavra de controle.

A monofásica se caracteriza pelo fato dos sinais de controle serem gerados num só ciclo da UCP.

A polifásica se caracteriza pelo fato dos sinais de controle serem gerados em dois ou mais ciclos de UCP.

A monofásica é vantajosa em certos casos onde o acesso à memória de controle é rápido. Se tal acesso for lento, poderá haver um aumento no tempo de execução.

A polifásica pode ter palavras de controle muito grandes, e, portanto, acarretar memórias de controle com muitos bits. No pior caso, cada sinal pode chegar a ter um ciclo próprio para ser gerado. Quanto maior a palavra, mais particular ela se torna e não pode ser usada em outros lugares nos microprogramas.

PARALELA E SERIAL

As microprogramações paralela e serial se distinguem quanto ao tempo em que a próxima palavra de controle é acessada da memória de controle.

Na paralela, a busca da próxima palavra de controle é sobreposta à execução da palavra de controle corrente.

Na serial, a busca da próxima palavra de controle é iniciada após completar-se a execução da corrente.

Na serial, se o tempo de acesso é muito menor do que o tempo de execução, o tempo total (acesso da palavra de controle mais execução) não é crítico. Levando em conta o tempo de acesso à memória principal, é menos crítico ainda.

Na paralela, caso o endereço da próxima palavra de controle esteja explícito na palavra corrente, a validade do uso desse tipo de microprogramação pode ser estimada pela seguinte fórmula (12):

$$\sum_{i=0}^{N} P_i T_i < \sum_{j=0}^{M} A_j$$

onde: P_i é a probabilidade de fracasso de ser feito um desvio do ponto i

T_i é o tempo perdido resultante do fracasso

A_j é o tempo de acesso

N é o número de desvios

M é o número de microinstruções.

Isto é, se num microprograma padrão o tempo total gasto na busca de palavras de controle sem sobreposição com o tempo de execução da anterior for menor que o total de tempos de acesso das palavras de controle com sobreposição, a microprogramação paralela ainda é justificada. Caso contrário, a serial seria preferível.

CODIFICADA E POUCO CODIFICADA

A microprogramação codificada ou pouco codificada se relaciona à estrutura da palavra de controle, quanto aos campos e codificação das micro-ordens.

A codificada se caracteriza pelo fato da palavra de controle possuir campos onde os sinais de controle mutuamente exclusivos são codificados em micro-ordens.

A pouco codificada seria o caso no outro extremo, onde cada campo corresponde a um sinal de controle, sendo que este campo praticamente não precisa ser decodificado.

Na codificada, as palavras de controle se assemelham mais à instruções de máquina e são mais fáceis de microprogramar.

Na pouco codificada, é necessário ter-se um conhecimento muito grande de todo o projeto lógico, e provavelmente, o próprio projetista seria o único capaz de microprogramar em tais condições [12].

As vantagens e desvantagens de cada tipo de microprogramação depende muito das características que se deseja para o sistema.

Para o sistema G-10 desenvolvido, que será detalhado no próximo capítulo, escolheu-se uma palavra de controle monofásica, paralela e codificada, apresentando porém alguns campos pouco codificados. Esta palavra de controle (no caso, uma microinstrução) foi dirigida para um minicomputador que possui um certo grau de paralelismo quanto às suas funções. Procurou-se aproveitar esse paralelismo, maximizando o número de campos com micro-ordens mutuamente exclusivos que agem em pontos diferentes da máquina simultaneamente.

III - EXEMPLO DE UMA UNIDADE DE CONTROLE MICROPROGRAMADA

Neste capítulo serão descritas as principais características do minicomputador G-10, apenas naquilo que se refere ao conjunto de elementos relevantes ao controle microprogramado. Não serão vistos aqui detalhes de hardware, técnica de implementação ou "timing". O sistema será apresentado apenas em termos de blocos funcionais.

III.1 CARACTERÍSTICAS FUNCIONAIS DO G-10

- O G-10 é um minicomputador de 16 bits, com uma memória principal endereçável até 64K palavras, com ciclo de 850ns.

- O número de instruções é de aproximadamente 130, todas microprogramadas, curtas (16 bits) ou longas (32 bits).

- O endereçamento é basicamente relativo, com ou sem indexação, com até dois níveis de endereçamento indireto.

- A memória é logicamente dividida em duas áreas: a de programa e a de dados. As localizações e tamanhos dessas áreas são determinados por dois registradores de base, BP (Base de Programa) e BD (Base de Dados) com endereços absolutos, e dois registradores de limites, LP (Limite de Programa) e LD (Limite de Dados). Essas áreas são protegidas por um esquema de proteção à memória. Normalmente, o endereço efetivo é calculado pela soma da base com o deslocamento indicado pela instrução, devendo este estar dentro da área permitida ao usuário.

- As instruções permitem a manipulação de pilhas e podem conter até dois endereços.

- Há dois tipos de canais de entrada e saída: o Canal Concentrador (que é programado por software, ao qual estão ligados os dispositivos de baixa velocidade) e o Canal Seletor (automático, para dispositivos de alta velocidade e acesso direto à memória).

- Além das interrupções externas pelos dispositivos de entrada e saída, existem as interrupções por falha de alimentação, instrução inválida, relógio interno, violação às partes protegidas da memória, painel, erro de paridade e "trace".

- Os modos de processamento são dois: Modo Usuário (algumas instruções privilegiadas são proibidas; o endereçamento está restrito à área de cada usuário) e Modo Supervisor (todas as instruções são permitidas; acesso integral à memória).

- O processador central dispõe dos seguintes elementos, como na figura III.1.

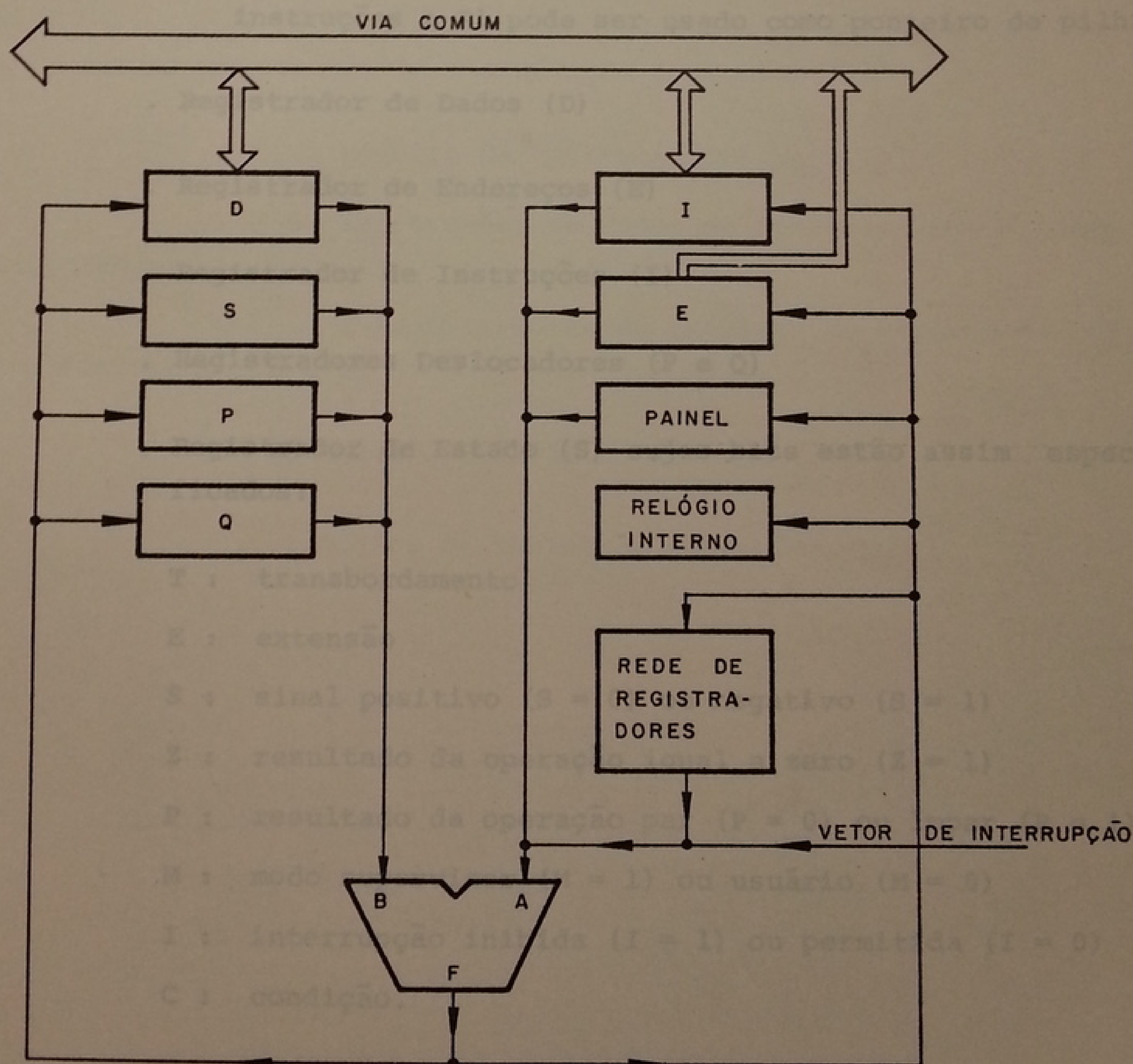


Figura III.1 - Fluxo de Dados do G-10

. Rede de 16 registradores, assim especificados:

- 4 registradores de trabalho (SP0, SP1, SP2, SP3), acessíveis apenas pela unidade de controle;

- 4 delimitadores de áreas (LD, LP, BD, BP). Esses registradores são acessíveis pela unidade de controle e pelo programador, através de instruções privilegiadas;

- 8 registradores de propósito geral (R0 a R7), acessíveis pela unidade de controle e pelo programador com instruções normais. Em particular, R0 é o contador de instruções e R1 pode ser usado como ponteiro de pilha.

. Registrador de Dados (D)

. Registrador de Endereços (E)

. Registrador de Instruções (I)

. Registradores Deslocadores (P e Q)

. Registrador de Estado (S) cujos bits estão assim especificados:

T : transbordamento

E : extensão

S : sinal positivo (S = 0) ou negativo (S = 1)

Z : resultado da operação igual a zero (Z = 1)

P : resultado da operação par (P = 0) ou ímpar (P = 1)

M : modo supervisor (M = 1) ou usuário (M = 0)

I : interrupção inibida (I = 1) ou permitida (I = 0)

C : condição.

O byte mais significativo é utilizado pelo programador por meio da instrução de Chamada de Supervisor.

. Unidade Lógica e Aritmética (ULA).

III.2 FLUXO DE INFORMAÇÕES NA UNIDADE DE CONTROLE

III.2.1 Descrição dos Elementos da Unidade de Controle

A Unidade de Controle é composta dos seguintes elementos:

. Registrador de Endereços do Controle (EC) com um circuito de seleção na sua entrada e somador "+1"

. Memória de Controle (MC)

. Registrador de Dados do Controle (DC) e decodificadores

. 2 Registradores de Endereço de Retorno de Micro-subrotina (S0 e S1)

. Contador (CNTD)

. Lógica de Testes de Condições

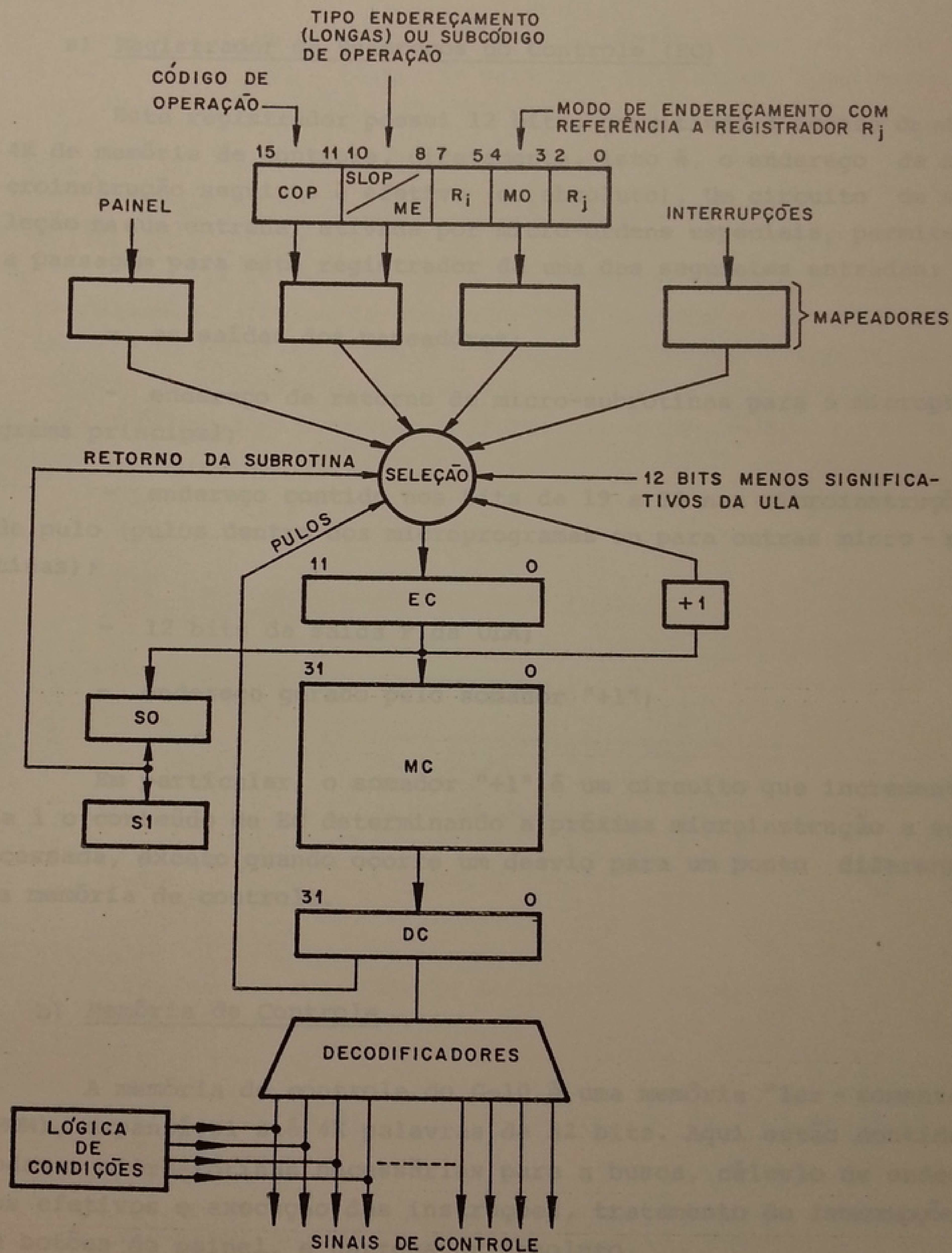


Figura III.2 - Fluxo do Controle

a) Registrador de Endereços do Controle (EC)

Este registrador possui 12 bits, permitindo o acesso de até 4K de memória de controle, diretamente, isto é, o endereço da microinstrução seguinte é efetivo (ou absoluto). Um circuito de seleção na sua entrada, ativada por micro-ordens especiais, permitem a passagem para este registrador de uma das seguintes entradas:

- as saídas dos mapeadores;

- endereço de retorno de micro-subrotinas para o microprograma principal;

- endereço contido nos bits de 19 a 30 nas microinstruções de pulo (pulos dentro dos microprogramas ou para outras micro-rotinas);

- 12 bits da saída F da ULA;

- endereço gerado pelo somador "+1";

Em particular, o somador "+1" é um circuito que incrementa de 1 o conteúdo de EC determinando a próxima microinstrução a ser acessada, exceto quando ocorre um desvio para um ponto diferente da memória de controle.

b) Memória de Controle

A memória de controle do G-10 é uma memória "ler-somente" (ROM), expansível até 4K palavras de 32 bits. Aqui estão contidas todas as micro-rotinas necessárias para a busca, cálculo de endereços efetivos e execução das instruções, tratamento de interrupções, de botões do painel, e carregador absoluto.

As palavras de 32 bits de cada microprograma são lidas seqüencialmente; esta seqüência pode ser alterada por uma micro-ordem de desvio, podendo retornar-se ou não ao ponto de desvio.

c) Registrador de Dados (DC) e Decodificadores

Após a leitura de uma microinstrução, esta é armazenada no Registrador de Dados do Controle (DC). Durante a sua execução, a memória de controle é novamente acessada, automaticamente, para a leitura da microinstrução seguinte, que só será armazenada no DC após o término da execução da anterior, o que se dá no fim de um ciclo.

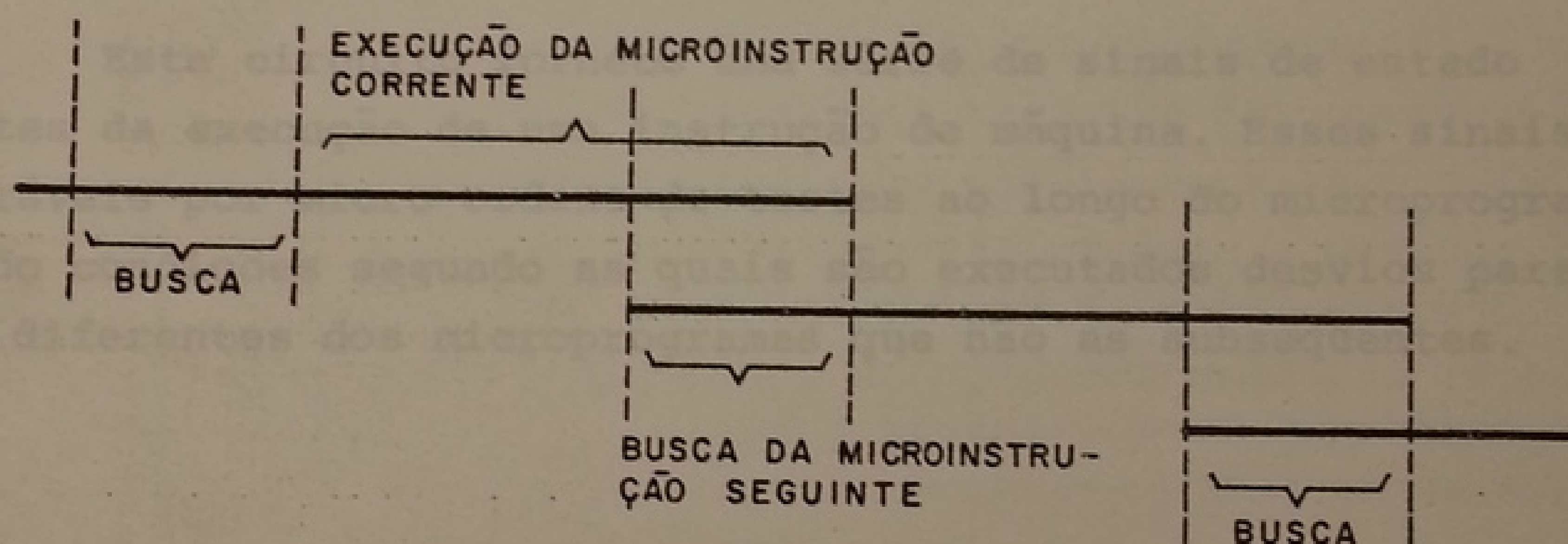


Figura III.3 - Ciclo de Controle (ou de UCP)

d) Registradores de Endereço de Retorno de Micro-Subrotinas (S0 e S1)

Os registradores S0 e S1, de 12 bits, guardam o endereço de retorno quando de uma chamada de micro-subrotina. Esses dois registradores, funcionando como pilha, permitem até dois níveis de chamada de micro-subrotina, no final das quais os endereços de

retorno são recuperados no EC, prosseguindo-se a execução do microprograma principal.

e) Contador (CNTD)

O contador de controle CNTD possui 4 bits e é carregável por uma micro-ordem especial ou pelo menos 4 bits menos significa da saída F da ULA. Ele é usado em todas as operações de contagem de até 16 vezes, como, por exemplo, no caso de deslocamentos. Também é usado para endereçar seqüencialmente os registradores da rede no fluxo de dados.

f) Lógica de Testes de Condições

Este circuito fornece uma série de sinais de estado decorrentes da execução de uma instrução de máquina. Esses sinais são testáveis por micro-ordens de testes ao longo do microprograma, gerando condições segundo as quais são executados desvios para pontos diferentes dos microprogramas que não as subseqüentes.

III.2.2 MAPEADOR

O mapeador pode ser considerado como uma interface entre o fluxo de dados e a unidade de controle. O elemento do fluxo de dados ao qual ele está conectado é o registrador de instruções (I).

Sua função básica é a de interpretar os códigos de operação das instruções, e associar aos mesmos um endereço na memória de controle correspondente à micro-rotina de execução da instrução. Esse endereço, quando selecionado, é colocado no registrador EC antes do acesso à memória de controle.

Há várias maneiras de executar-se um mapeamento a partir do código de operações. Para tanto, é necessário conhecer-se os formatos das instruções, principalmente os campos que devem ser interpretados. Os formatos gerais das instruções do G-10 são os seguintes:

Instruções curtas:

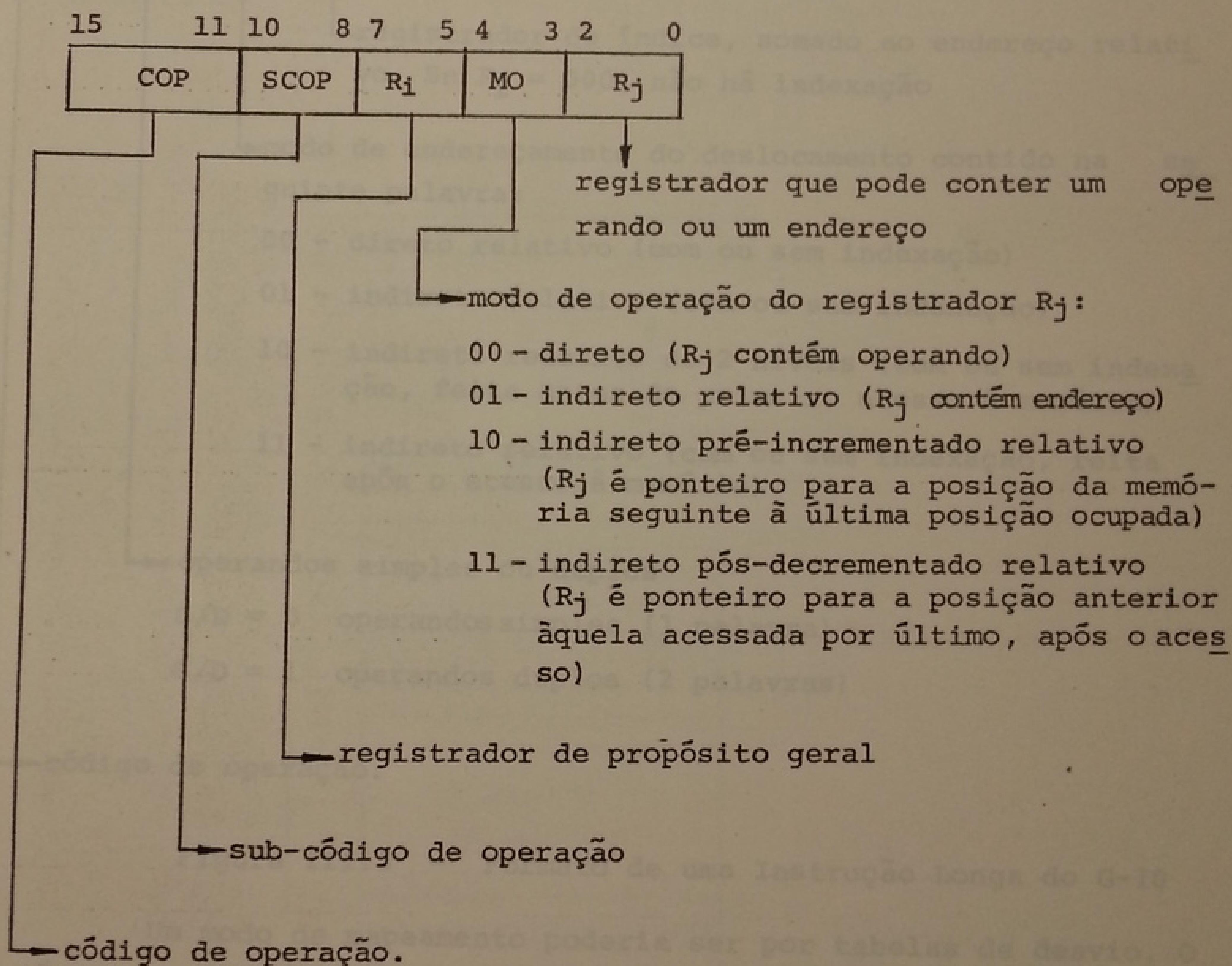


Figura III.4 - Formato de uma Instrução Curta do G-10

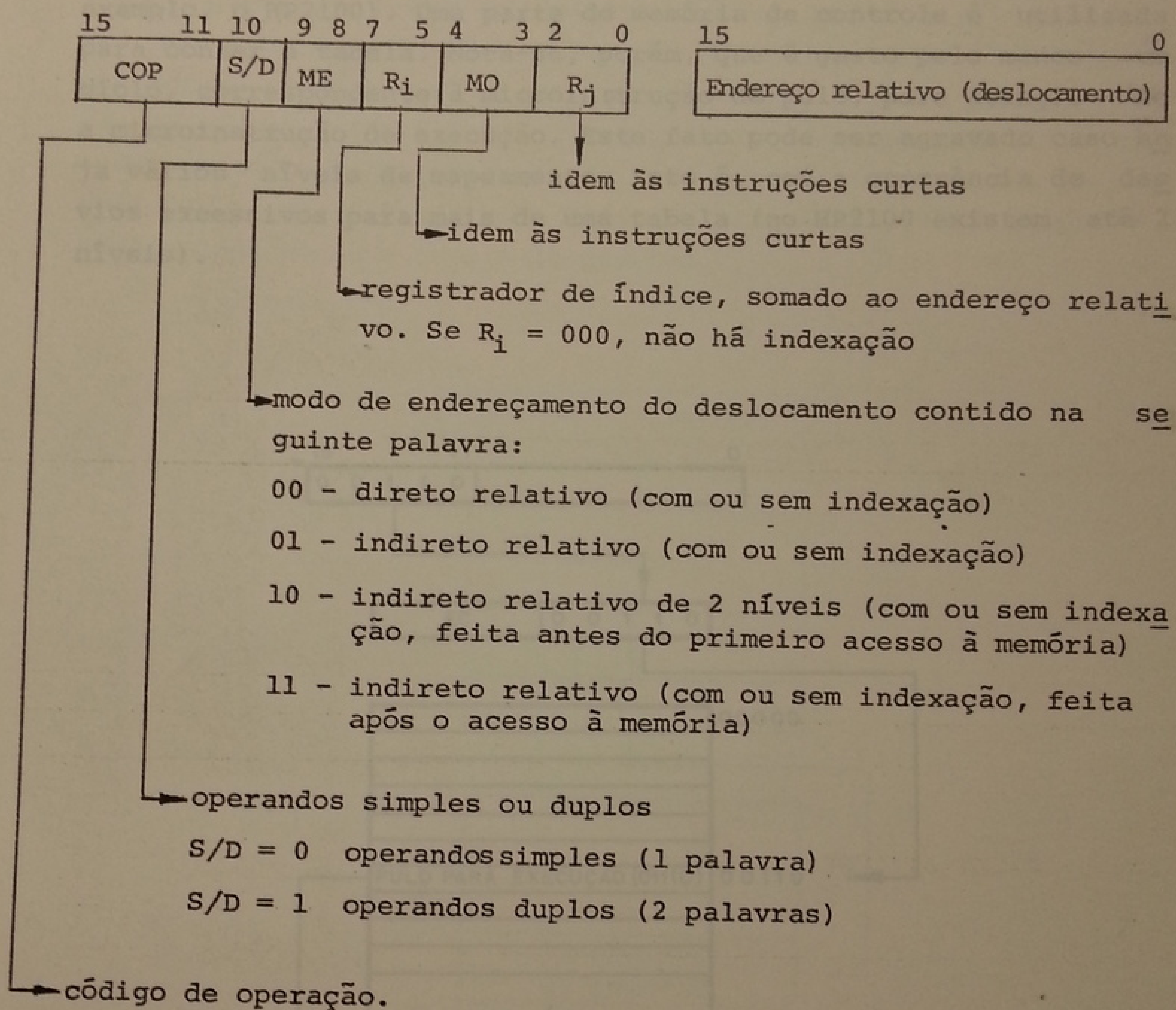


Figura III.5 - Formato de uma Instrução Longa do G-10

Um modo de mapeamento poderia ser por tabelas de desvio. O número em código binário do código de operação poderia ser o endereço do ponto de entrada de uma tabela de ponteiros, sendo selecionado assim o endereço da micro-rotina de execução propriamente dita.

Esse tipo de mapeamento é usado em alguns sistemas (por exemplo, o HP2100). Uma parte de memória de controle é utilizada para conter a tabela. Nota-se, porém, que é gasto pelo menos um ciclo, correspondente à microinstrução de pulo, para atingir-se a microinstrução de execução. Este fato pode ser agravado caso haja vários níveis de mapeamento, isto é, com a ocorrência de desvios excessivos para mais de uma tabela (no HP2100 existem até 2 níveis).

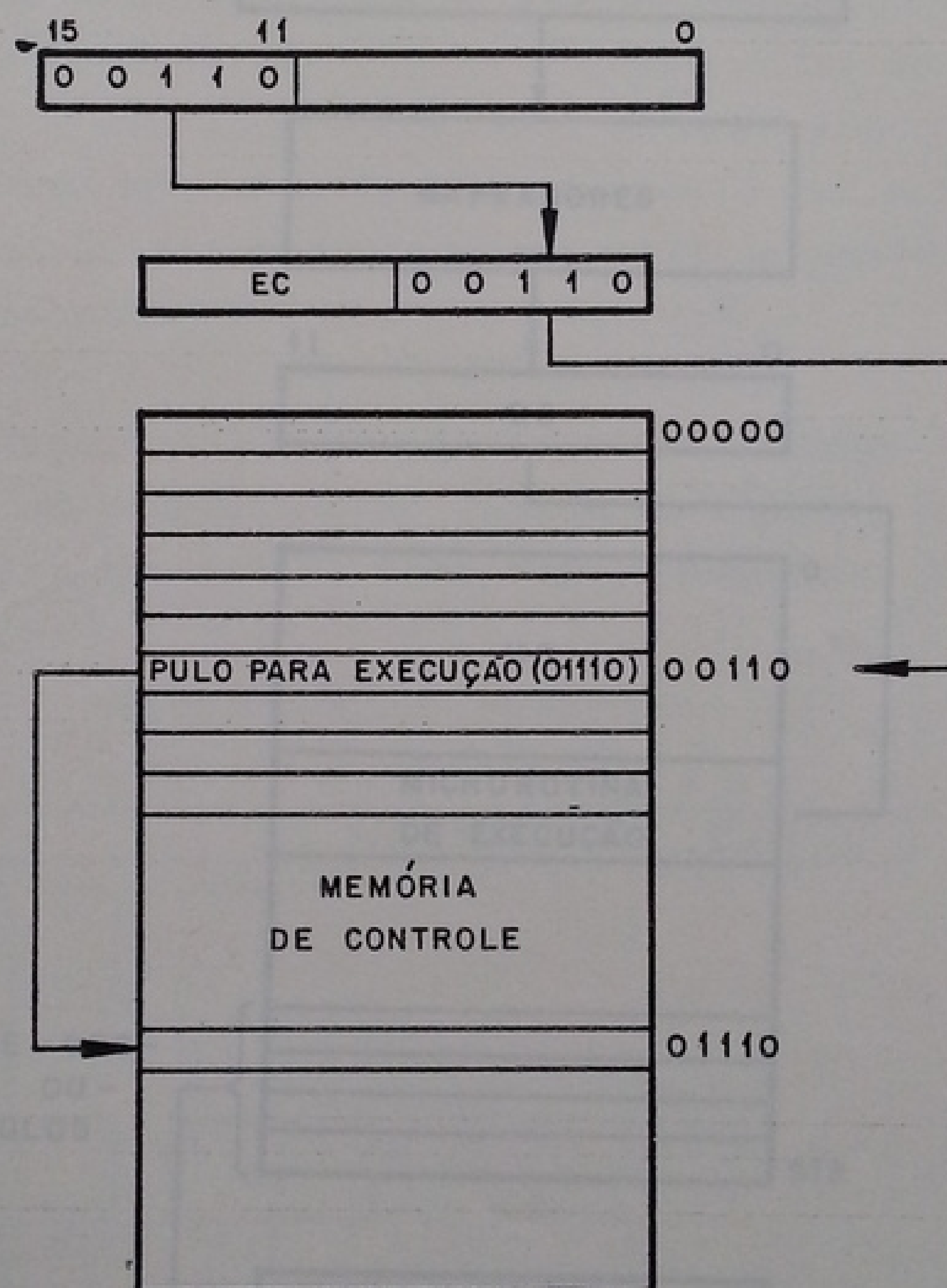


Figura III.6 - Mapeamento por Tabelas de Desvio

No G-10, como será visto, o mapeamento é feito por memórias ler-somente, cujas entradas estão ligadas a bits determinados do registrador de instruções, e cuja saída fornece o endereço real da micro-rotina de execução, automaticamente, desde que ela se encontre nas 512 primeiras palavras de controle, onde estão localizados os microprogramas do conjunto básico de instruções. Qualquer micro-rotina situada além das primeiras 512 palavras, será acessada por uma tabela de desvios.

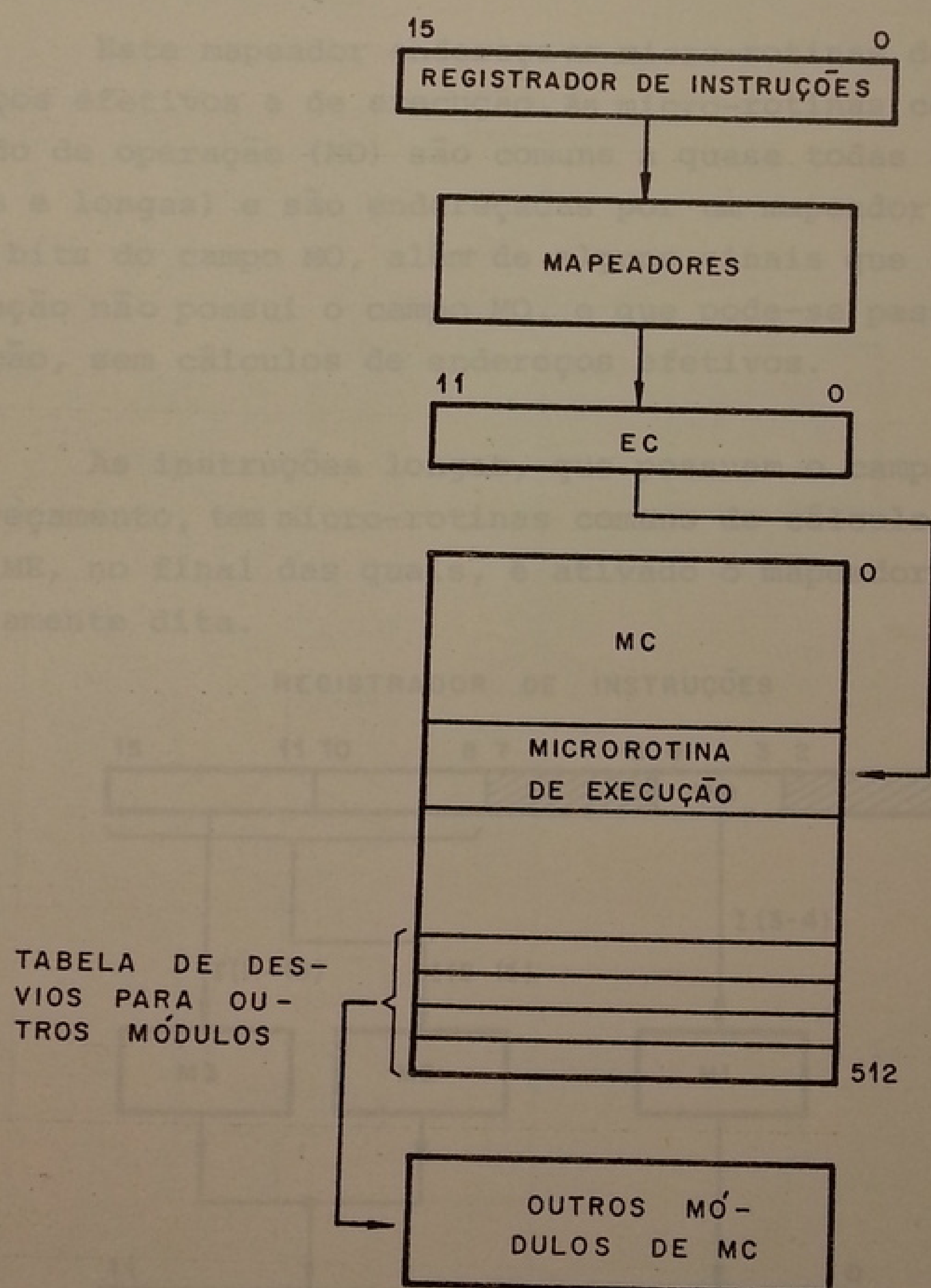


Figura III.7 - Tipo de Mapeamento do G-10

São três os mapeadores do G-10, todos implementados com memórias ler-somente (ROM):

- Mapeador de instruções
- Mapeador do painel
- Mapeador de interrupções.

Mapeador de instruções

Este mapeador endereça as micro-rotinas de cálculo de endereços efetivos e de execução. As micro-rotinas correspondentes ao modo de operação (MO) são comuns a quase todas as instruções (curtas e longas) e são endereçadas por um mapeador cujas entradas são os bits do campo MO, além de alguns sinais que indicam se a instrução não possui o campo MO, e que pode-se passar à fase de execução, sem cálculos de endereços efetivos.

As instruções longas, que possuem o campo ME de modo de endereçamento, tem micro-rotinas comuns de cálculo de endereço efetivo ME, no final das quais, é ativado o mapeador de execução propriamente dita.

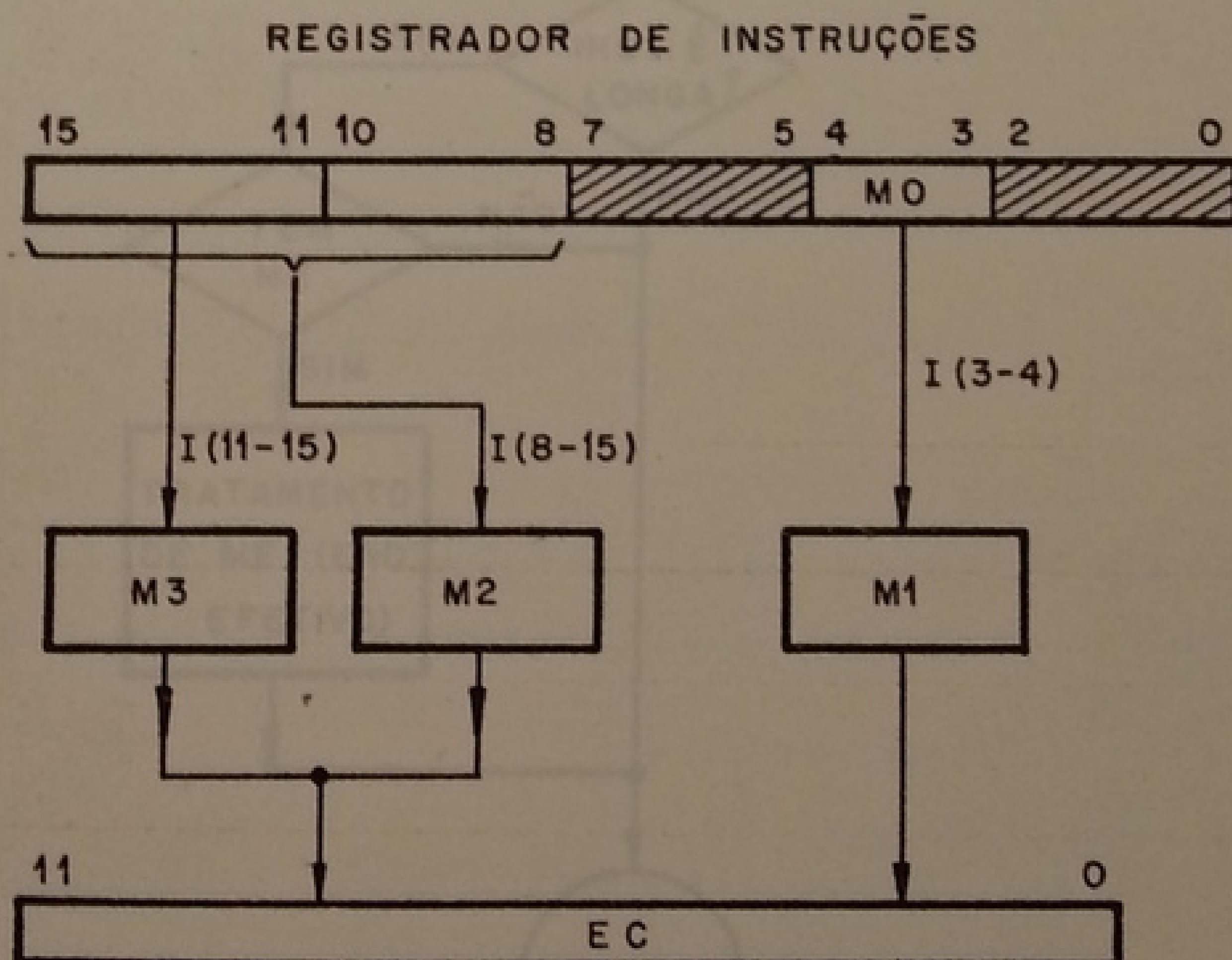


Figura III.8 - Mapeador de Instruções

Logo após a instrução estar no registrador I, o mapeador M1 endereça uma micro-rotina que trata MO. A seguir, é acionado o mapeador M2 que endereça uma micro-rotina de tratamento de ME nas instruções longas ou diretamente para uma micro-rotina de execução (instruções curtas). Por último (após o tratamento de ME), é acionado o mapeador 3, no caso das instruções longas, para a execução das mesmas.

O diagrama de blocos do funcionamento seqüencial dos mapeadores é o seguinte:

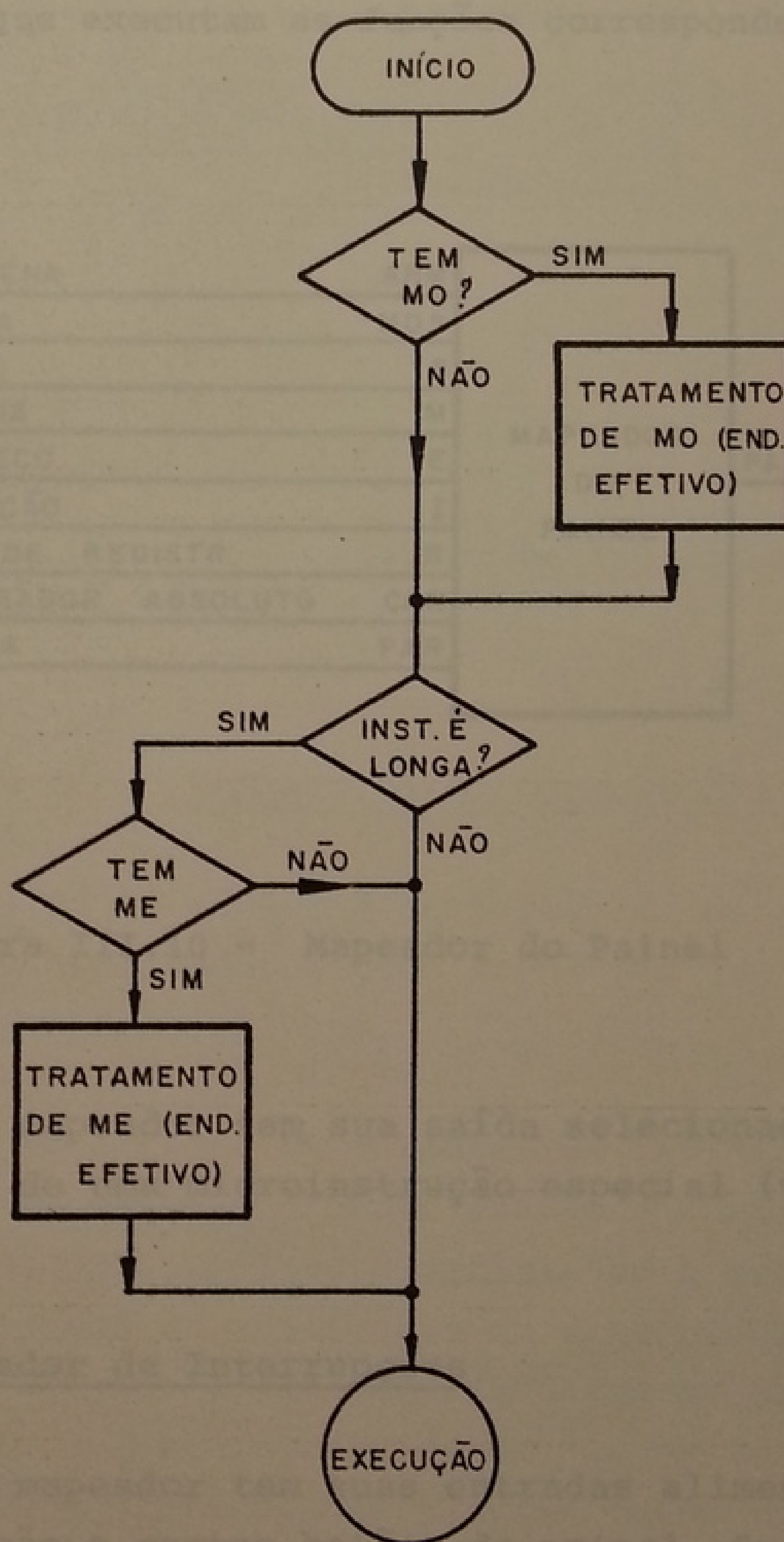


Figura III.9 - Seqüência de Mapeamento

O mapeador M1 tem suas saídas selecionadas para o registrador EC através de uma micro-ordem especial. Os mapeadores M2 e M3, sendo mutuamente exclusivos quanto ao seu uso, possuem suas saídas ligadas, selecionáveis também por micro-ordens especiais (vide apêndice A).

Mapeador do Painel

Esse mapeador tem suas entradas alimentadas por um conjunto de botões do painel, e sua finalidade é a de endereçar microprogramas que executam as funções correspondentes a cada botão.

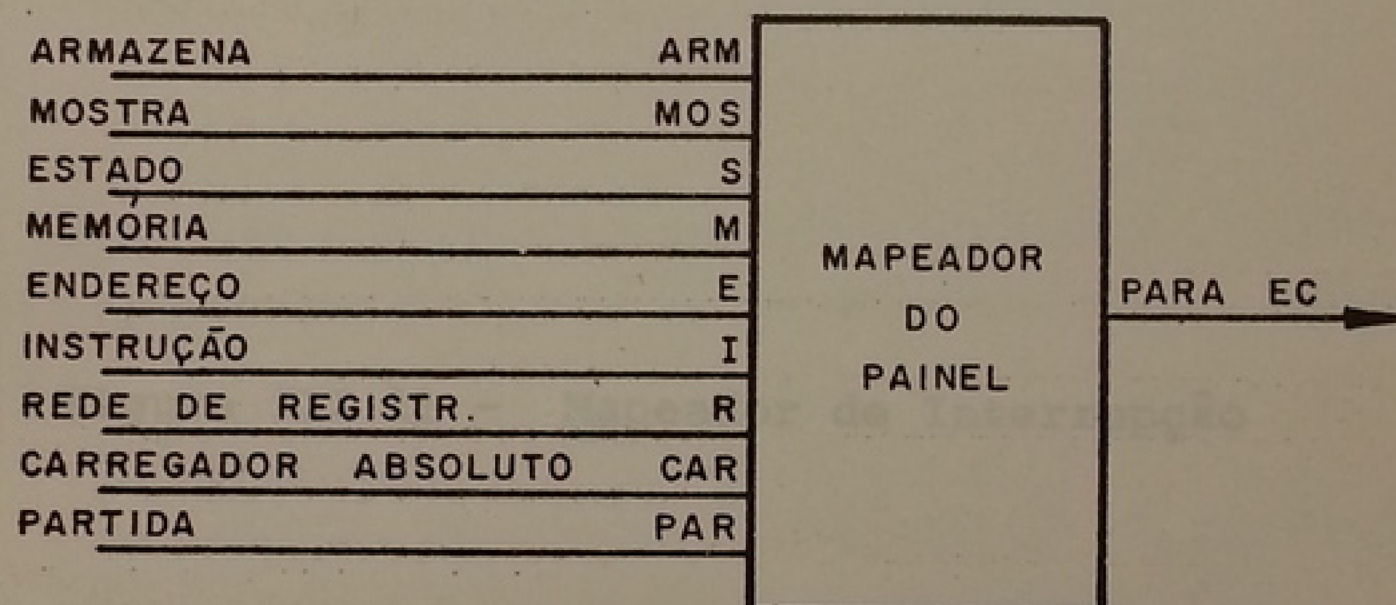


Figura III.10 - Mapeador do Painel

Esse mapeador tem sua saída selecionada para o registrador EC por meio de uma microinstrução especial (vide apêndice A).

Mapeador de Interrupções

Esse mapeador tem suas entradas alimentadas pelo circuito de interrupção e certos botões do painel. Sua finalidade é a de endereçar uma micro-rotina de tratamento de cada interrupção que possa ocorrer.

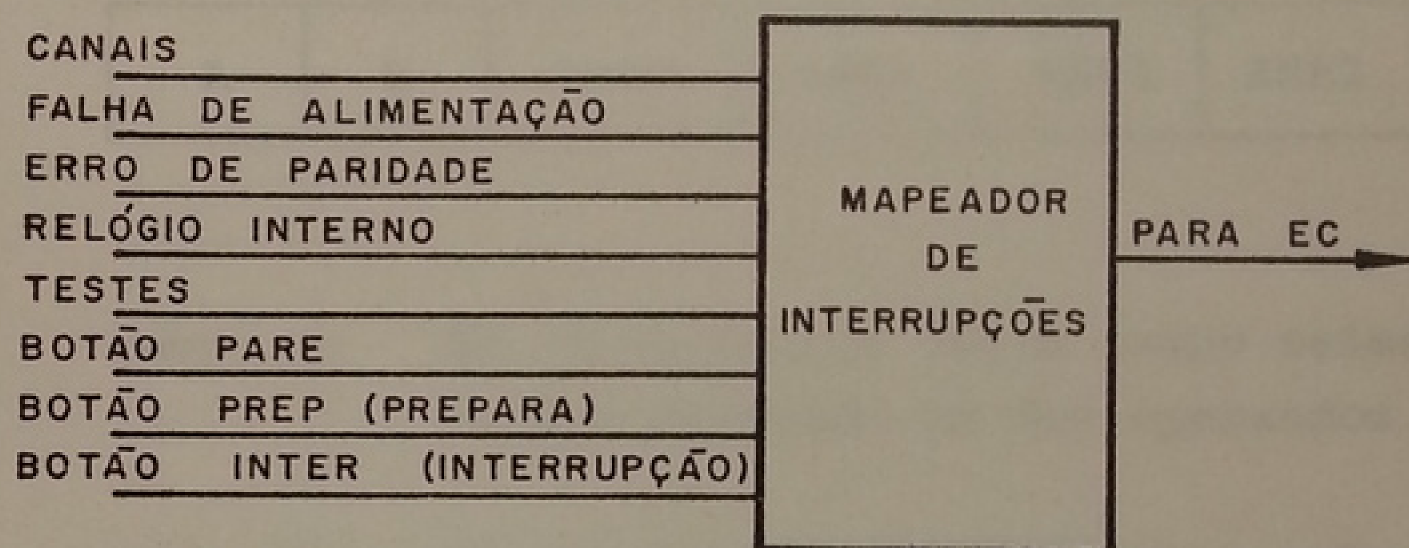


Figura III.11 - Mapeador de Interrupção

III.2.3 - FORMATO DA MICROINSTRUÇÃO

Antes de se passar à descrição da microinstrução atual do G-10, convém esclarecer alguns problemas que surgiram quando da definição da microinstrução.

Para se definir os formatos de microinstruções em geral, é preciso ter-se um conhecimento bastante aprofundado do hardware da máquina, e principalmente definir-se o que se deseja obter da máquina em termos de eficiência no desempenho. Deve-se levar em conta quais funções podem ser efetuadas em paralelo, e quais as mais críticas, isto é, quais devem ser executadas com maior rapidez. Conforme a variação que algumas características possam sofrer, esta far-se-á sentir na definição da microinstrução pela adição

de um novo campo ou uma nova micro-ordem num dos campos.

A primeira idéia para a definição da microinstrução englobava os seguintes campos:

32 ... 0

A	B	FUNC	ARMA	ESP1	ESP2	IMED

Campo A: As micro-ordens deste campo selecionavam o registrador contendo um dos operandos.

Campo B: As micro-ordens deste campo selecionavam o outro operando.

Tanto em A como em B poderiam ser selecionados praticamente quaisquer registradores: de propósito geral, de trabalho ou especiais.

Campo FUNC: Este campo continha as micro-ordens de operações da ULA ou de deslocamento.

Campo ARMA: Especificava o destino do resultado da operação da ULA ou de deslocamento.

Campo ESP1: Este campo continha micro-instruções de pulos não condicionados, acessos à memória além de outras micro-ordens auxiliares.

Campo ESP2: Este campo continha micro-ordens de pulos condicionais (condições no campo IMED), micro-ordens de geração de máscaras (máscaras no campo IMED) e leitura ou escrita nos registradores da rede.

Cabe salientar que na arquitetura antiga, os registradores da rede não tinham acesso direto a uma das entradas da ULA. Seus conteúdos deviam ser inicialmente armazenados nos registradores P ou Q por uma via que não a ULA.

Campo IMED: Continha as máscaras, constantes, condições e micro-ordens de transferência de dados entre registradores da rede e os registradores P e Q.

Com a elaboração de vários microprogramas, observaram-se os seguintes fatos:

a) Tanto o campo ESP1 como o ESP2 possuíam micro-ordens iguais, a menos delas serem condicionais. Observam-se que esta formulação poderia ser compactada, isto é, adotar-se apenas o uso das micro-ordens de pulo condicionais, sendo que, se nenhuma condição fosse especificada, elas seriam executadas incondicionalmente.

b) O campo ESP2 era freqüentemente usado para leitura ou escrita de um registrador da rede para o registrador P ou Q, após o que o operando é manipulado pela ULA. Esse problema foi eliminado, efetuando-se a ligação direta da rede de registradores à entrada A da ULA, e portanto qualquer operando da rede pode ser manipulado diretamente pela ULA. O campo ESP2 foi eliminado, sendo substituído pelo atual campo COND para a especificação de condições.

c) Qualquer acesso à memória principal acarretava um tempo de espera por parte da UCP, até que o dado lido estivesse no registrador de dados D. Esta espera não permitia nenhum processamento paralelo na UCP que não envolvesse o dado acessado. Esta característica foi otimizada pela criação de um campo (campo R) de 1 bit que delimita uma seqüência de operações a serem realizadas na UCP em paralelo com um acesso à memória; somente após a execução de tal seqüência de operações, a UCP permanece em estado de espera pelo dado. Na maioria das vezes, esse tempo de espera pode-se

tornar insignificante, quando é bem aproveitado esse tipo de paralelismo em microprograma.

d) Todos os campos eram codificados para gerar os sinais de controle, o que implicava em um circuito relativamente grande para decodificação. Com um remanejamento dos campos, porém, conseguiu-se separar as micro-ordens cuja decodificação era indispensável, das micro-ordens que exerciam praticamente controle direto sobre os elementos lógicos da UCP. Exemplos desse últimos campos na microinstrução atual é o campo R, campo A, MUXE, END, que serão vistos a seguir.

A microinstrução do G-10 é constituída de 32 bits divididos em 10 campos contendo as várias micro-ordens cuja decodificação gera sinais de controle que acionarão os circuitos convenientemente do Processador Central e da própria Unidade de Controle, numa seqüência tal que os algorítmos sejam corretamente executados.

O formato da micro-instrução é o seguinte:

31	30 29	28 26	25 20	19 18	17 16	15 12	11 7	6 4	3 0
R	A	B	FUNC	MUXE	D	ARMA	ESP	COND	END

Cada campo possui um grupo de bits codificados que determinam a operação a ser efetuada pelo controle. Cada código desses grupos corresponde a uma micro-ordem.

As operações gerais especificadas em cada campo são as seguintes:

Campo R: As micro-ordens deste campo controlam o relógio central no sentido de permitir a geração dos ciclos de controle ou de inibí-los. São usadas por ocasião de um acesso à memória prin

cipal ou à Via Comum de Comunicação, a fim de parar o processamento de micro-instruções subseqüentes até que o dado acessado esteja seguramente correto no registrador de dados do Processador Central.

Campo A — Este campo contém as micro-ordens que selecionam o registrador do Fluxo de Dados que terá acesso à entrada A da ULA.

Estas micro-ordens funcionam em conjunto com as dos campos MUXE e END.

Campo B — Este campo contém as micro-ordens que selecionam o registrador do Fluxo de Dados que terá acesso à entrada B da ULA.

Campo FUNC — Este campo contém os códigos das funções lógicas e aritméticas da ULA, ou das operações de deslocamento efetuadas pelos registradores deslocadores P e Q, com ou sem atualização dos bits do registrador de estado S do processador.

Campo MUXE — As micro-ordens deste campo controlam a seleção e endereçamento dos registradores da rede.

Elas funcionam em conjunto com as micro-ordens dos campos A e END.

Campo D — Este campo contém as micro-ordens que determinam o sentido de deslocamento ou a carga dos registradores deslocadores P e Q.

Estas micro-ordens funcionam em conjunto com as do campo FUNC.

Campo ARMA — As micro-ordens deste campo são responsáveis pela seleção do registrador onde será armazenado o resultado da operação realizada no ciclo de controle corrente. Este resultado é provindo da saída F da ULA.

Estas micro-ordens eventualmente funcionam em conjunto com as dos campos MUXE e END, quando for selecionado um dos registradores da rede.

Campo ESP — Este campo contém os códigos das micro-ordens especiais que efetuam pulos para micro-rotinas diferentes, acessos à memória ou à via comum, testes de condições, controle sobre o circuito de seleção para o registrador EC, e carga imediata do contador CNTD.

Estas micro-ordens eventualmente funcionam em conjunto com as do campo COND, e do campo END, quando for determinada a carga do contador CNTD.

Campo COND — Este campo contém as condições que devem ser testadas pelas micro-ordens condicionais do campo ESP. Estas condições só podem ser testadas por micro-programas e nunca por instruções de máquina. A condição é dita verdadeira se ela for igual a 1.

Campo END — Este campo contém o código dos registradores da rede, vetor de interrupção e registrador de chaves do painel endereçáveis pelas micro-ordens dos campos A e MUXE.

As micro-ordens de cada campo estão descritas no apêndice A.

IV - MICROPROGRAMAÇÃO DAS INSTRUÇÕES

IV.1 FASES DA MICROPROGRAMAÇÃO DE UMA INSTRUÇÃO

Toda instrução passa por três fases durante a sua execução:

- Fase de Busca
- Fase de Cálculo de Endereço Efetivo
- Fase de Execução.

Normalmente, após cada fase é acionado um dos mapeadores que endereça a memória de controle para a posição inicial de uma micro-rotina pertencente à fase seguinte. Assim, após a fase de busca, o mapeador M1 (endereçamento MO) é acionado: é iniciada a fase de cálculo de endereço efetivo. Após terminada a execução do cálculo de endereço efetivo MO, o mapeador M2 é acionado para o cálculo de endereço efetivo ME ou eventualmente uma rotina de execução. Após o tratamento de ME, o mapeador M3 é acionado, iniciando-se assim a fase de execução.

FASE DE BUSCA

A micro-rotina de busca de instrução ocupa o endereço zero na memória de controle e é comum a todas as instruções. O endereço da instrução acessada é obtido a partir do contador CI e a base de programa.

Esta micro-rotina acessa a área de programa da memória, uma ou duas vezes, dependendo da instrução ser curta ou longa.

Ao término desta fase, uma micro-ordem especial (FOR1) aciona o mapeador M1, iniciando-se a fase de cálculo de endereço efetivo.

FASE DE CÁLCULO DE ENDEREÇO EFETIVO

Para cada tipo de endereçamento existe uma micro-rotina correspondente para o cálculo de endereço efetivo dos operandos, referentes tanto à área de dados como a de programa.

Logo após a fase de busca, é feito o cálculo de endereço efetivo MO. A micro-ordem FOR2 no final dessas micro-rotinas ativa o mapeador M2 para uma micro-rotina tratadora de ME. A micro-ordem EEX usada em conjunto com FOR2 mapeia uma micro-rotina de execução.

As condições finais de cada micro-rotina de endereçamento são padronizadas para qualquer micro-rotina de execução. Por exemplo, sabe-se que o endereço efetivo MO se encontra sempre no registrador SP1 para qualquer instrução, assim como o registrador E contém sempre o endereço efetivo ME.

FASE DE EXECUÇÃO

Esta fase abrange as operações de preparação de dados, execução propriamente dita, finalização e teste de interrupção.

Preparação de dados. Este passo consiste no acesso à memória para a busca de operandos. Este será sempre armazenado no registrador SP2. Se não houver necessidade de preparação de dados, passa-se diretamente ao passo seguinte.

Execução propriamente dita. Este passo consiste no processamento dos algorítmos implementados, utilizando os operandos do passo anterior.

Finalização. Este passo consiste em armazenar os resultados da operação no registrador ou posição de memória apropriada, conforme o modo de operação (MO) ou endereçamento (ME). Com a microinstrução FIM, está implícito um teste de um pedido de interrupção.

<u>Teste de interrupção</u>. As interrupções são normalmente atendidas ao término de uma instrução, exceção feita às interrupções do tipo instrução inválida e violação às partes protegidas da memória, que são testadas por micro-ordens condicionais.

No caso de não ser detetada nehuma interrupção, volta - se para a fase de busca da próxima instrução do programa corrente. Caso contrário, é feito um desvio para uma micro-rotina tratadora da interrupção, que além de salvar informações sobre o estado do processador na memória principal e acionar o Modo Supervisor, endereça a primeira instrução do programa de atendimento da interrupção, residente na memória.

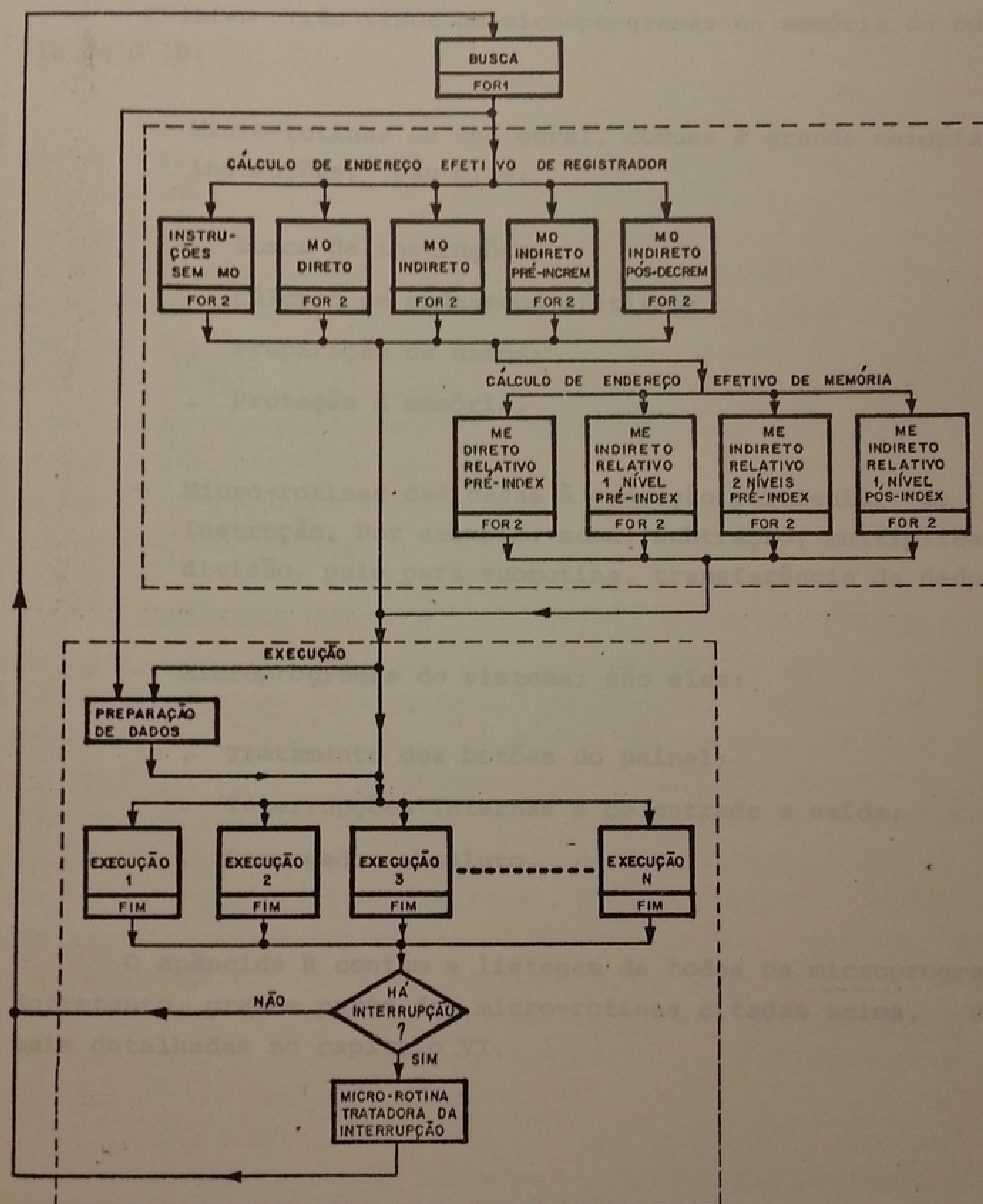


Figura IV.1 - Fases das Instruções

IV.2 - TIPOS DE MICROPROGRAMAS

Existem três tipos de microporgramas na memória de controle do G-10:

- Micro-rotinas de uso geral, comuns à grande maioria das instruções; são elas:

  - Busca de instruções;
  - Cálculo de endereços efetivos;
  - Preparação de dados;
  - Proteção à memória.

- Micro-rotinas dedicadas à execução particular de cada instrução. Por exemplo: soma, subtração, multiplicação, divisão, pulo para subrotina, transferência de dados, etc.

- Microprogramas do sistema; são eles:

  - Tratamento dos botões do painel;
  - Interrupções internas e de entrada e saída;
  - Carregador absoluto.

O apêncide B contém a listagem de todos os microprogramas. Entretanto, grande parte das micro-rotinas citadas acima, estão mais detalhadas no capítulo VI.

IV.3 DEPURAÇÃO DOS MICROPROGRAMAS

Estes microprogramas passaram por várias fases durante seu desenvolvimento.

Inicialmente, foram construídos os diagramas de blocos correspondentes aos algorítmos, passando-se a seguir pela codificação e perfuração de cartões. Os microprogramas assim codificados foram montados por um micromontador desenvolvido para o sistema HP2116. Este micromontador produz, além da listagem, a sua versão em códigos hexadecimal e binário, em fita de papel perfurada, cujo conteúdo é exatamente a informação que estaria contida na memória de controle. O micromontador também fornece uma listagem de erros que porventura possam ter ocorrido durante a montagem: códigos errados de micro-ordens, codificação fora dos campos, utilização simultânea de micro-ordens mutuamente exclusivas, rótulos de microrotinas não definidos, etc.

O estágio seguinte foi a depuração lógica dos microprogramas. Este passo foi realizado em parte através de uma simulação em nível de "gates" da arquitetura do sistema, também feita no sistema HP2116. Este processo consistia em fazer-se executar o microprograma já montado na arquitetura simulada, e observar-se, durante certos intervalos de tempo, o comportamento do fluxo de informações entre os blocos funcionais da arquitetura. Além da descoberta de erros nos microprogramas, pode-se também constatar certas falhas quanto à implementação da arquitetura, como por exemplo, problemas com "timing", sincronização, elementos lógicos produzindo funções erradas, etc. O microprogramador reportava essas conclusões à equipe de hardware, e interagia com a mesma para a modificação dos circuitos lógicos.

Após essa depuração feita por software, passava-se a executar o conjunto de microprogramas na própria máquina. Inicialmente foram depurados os microprogramas um por vez, e quando da constatação que todos funcionavam individualmente, fez-se a integração de todos, tendo sido tornado possível o processamento de pequenos programas com várias instruções.

Finalmente, os microporgramas já depurados e corretos passaram por uma última montagem, gerando-se uma fita de papel contendo as informações de controle que eram carregadas na memória ler-escrever através de um periférico (TTY ou leitora de fita).

V - OTIMIZAÇÃO DA MICROPROGRAMAÇÃO

Uma vez definidas as operações que um microprograma deve executar, operações estas representadas por uma seqüência de microinstruções, torna-se necessário fazer-se uma verificação da eficiência da microprogramação, pois este é um dos pontos principais que determinam a eficiência do desempenho da máquina. São fundamentais os seguintes fatores:

- Espaço de memória de controle
- Tempo de execução.

O espaço ocupado pelos microprogramas determina o tamanho da memória de controle necessária. Quanto maior for este, mais caro se torna o projeto da unidade de controle, por ser necessária a utilização de grande número de pastilhas de ROM, podendo chegar a ocupar uma área muito grande no circuito impresso.

O tempo de execução de um microprograma determina o tempo de uma instrução, que deve ser o mais rápido possível. A transição de uma fase para outra deve ser rápida, e o número de microinstruções deve ser o mínimo possível, com máximo aproveitamento do paralelismo da UCP.

V.1 OTIMIZAÇÃO EM TEMPOS DE ESPAÇO

a) Uso de Micro-Subrotinas

Uma primeira providência é avaliar o número de vezes que uma certa seqüência de microinstruções iguais se repete ao longo de um microprograma, ou mesmo num conjunto de microprogramas. Se este número for elevado, certamente seria bastante econômico organizar estas microisntruções de tal forma a constituirem uma micro-subrotina facilmente chamada pelos microprogramas.

b) Micro-Subrotinas Mapeadas

Uma outra técnica de otimização consiste na observação do número de vezes que uma micro-subrotina é chamada. Se tal número for razoável (o conceito do que seja um número "razoável" depende da estrutura da microprogramação dos procedimentos. No G-10, por exemplo, esse número pode atingir 4 ou 5 vezes) pode-se fazer a chamada da micro-subrotina por uma microinstrução de pulo.

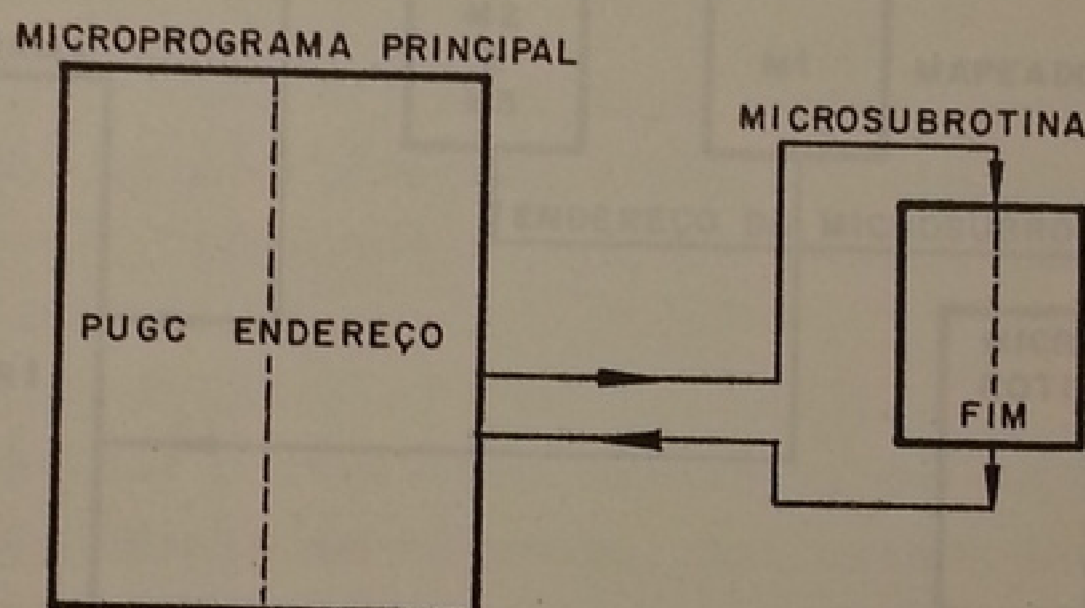


Figura V.1 - Chamada Simples de Micro-Subrotina

Se, porém, o número de vezes que uma micro-subrotina é chamada for muito elevado (no projeto mais de 5 ou 6 vezes), pode-se lançar mão da técnica de mapeamento, que consiste no seguinte: no transcorrer do microprograma principal, quando houver necessidade de chamar a micro-subrotina, pode-se incluir na última microinstrução uma micro-ordem de mapeamento que irá forçar, através do mapeador, o endereço de desvio para tal micro-subrotina. No final desta apareceria uma outra micro-ordem de mapeamento para o retorno ao microprograma principal, ou eventualmente a outro microprograma.

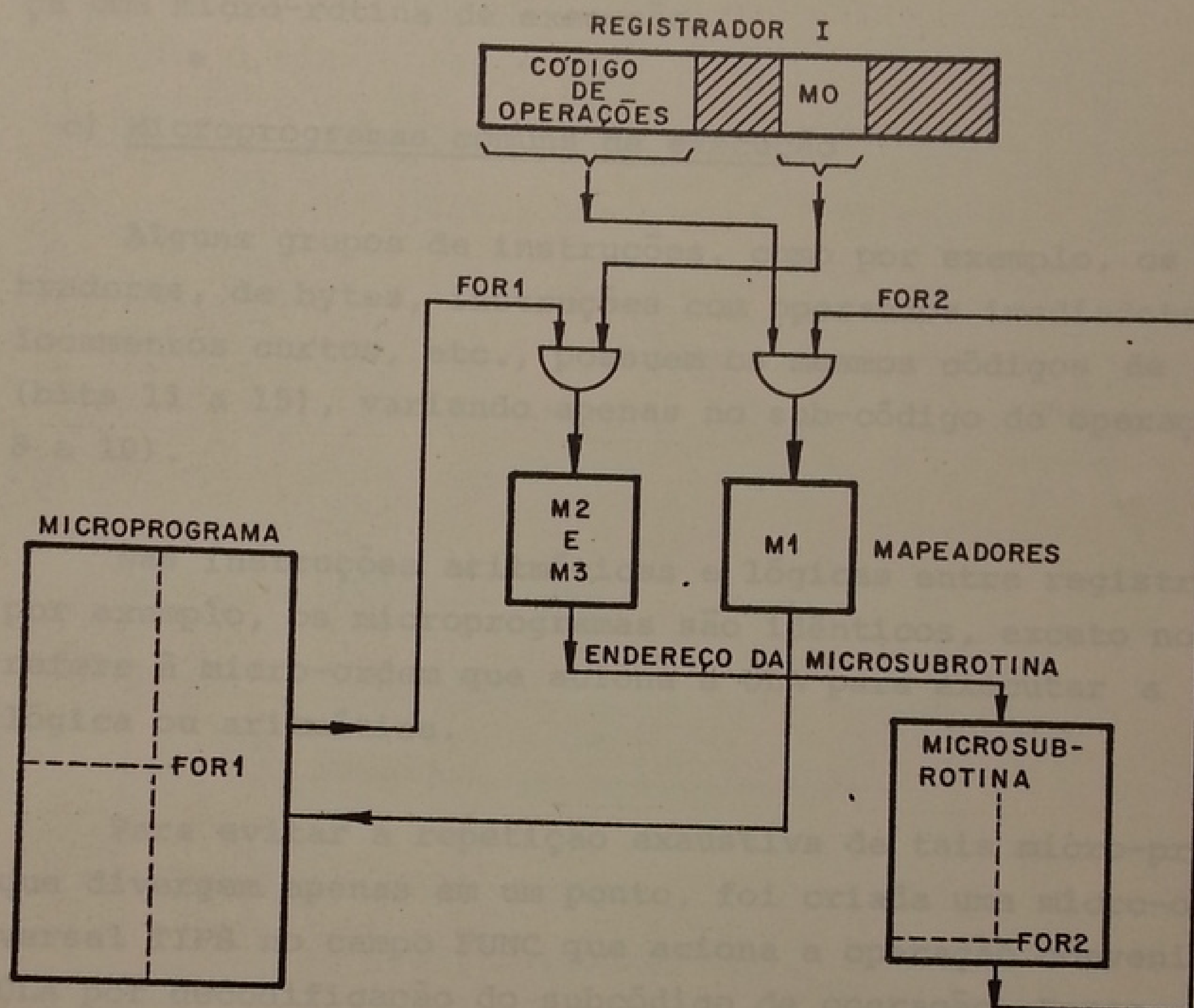


Figura V.2 - Micro-Subrotinas Mapeadas

Na figura acima, a micro-ordem FOR1 chama a micro - subrotina. O mapeador M1 gera o endereço da micro-subrotina a qual é executada a seguir. No final desta, a micro-ordem FOR2 mapeia o endereço de retorno para o microprograma.

Esta característica é bastante utilizada no G-10. Por exemplo, a fase de cálculo de endereço efetivo e a fase de execução das instruções constam de microprogramas consecutivos chamados por mapeamento direto.

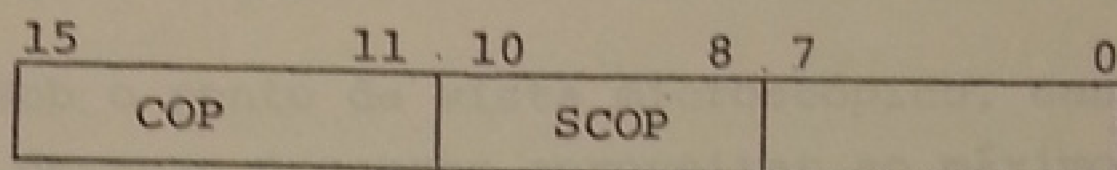
Outro exemplo é o caso das instruções imediatas curtas. Logo após a busca, a micro-ordem FOR1 endereça uma micro-rotina de preparação de dados. No final destas, a micro-ordem FOR2 endereça uma micro-rotina de execução.

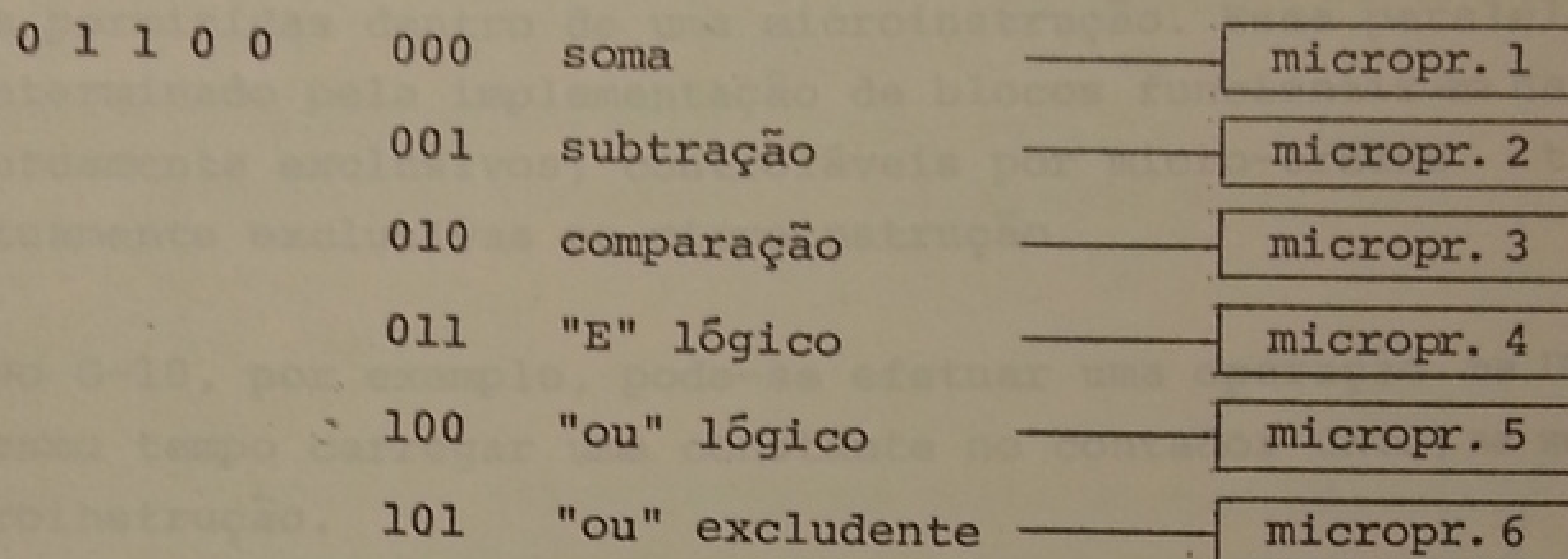
c) Microprogramas comuns de execução

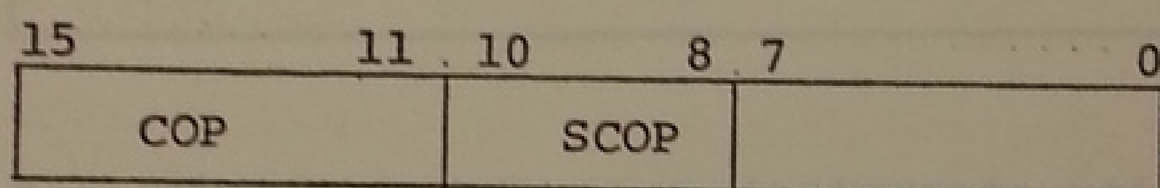
Alguns grupos de instruções, como por exemplo, os de registradores, de bytes, instruções com operandos imediadots, de deslocamentos curtos, etc., possuem os mesmos códigos de operação (bits 11 a 15), variando apenas no sub-código de operação (bits 8 a 10).

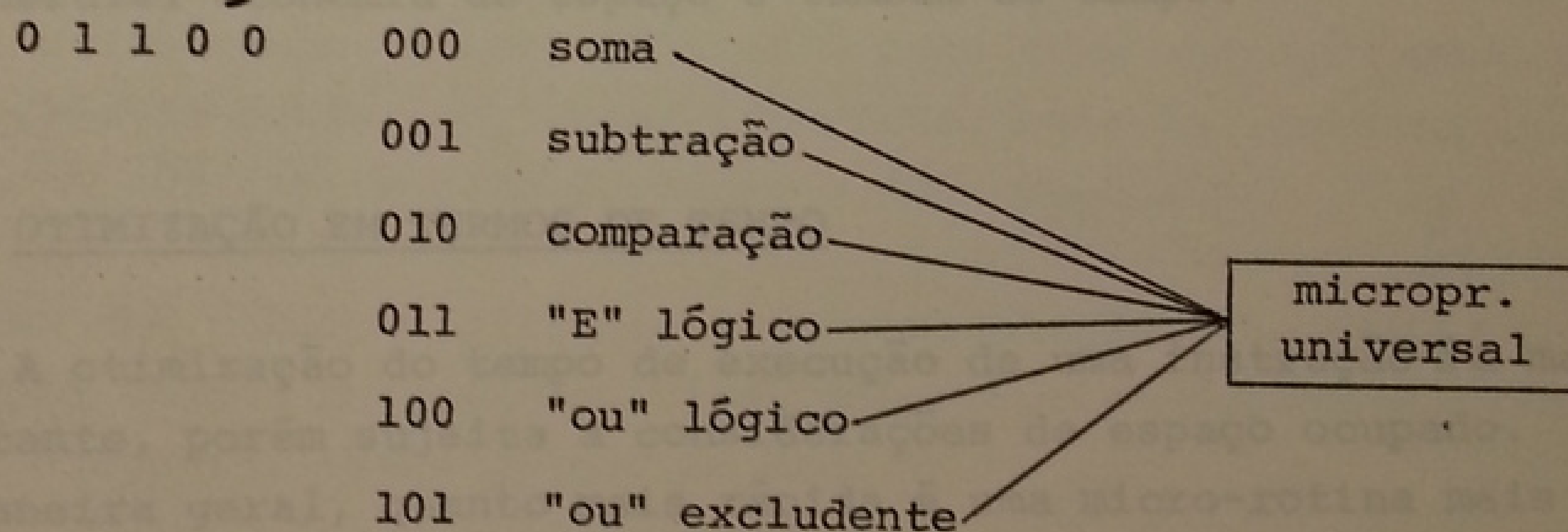
Nas instruções aritméticas e lógicas entre registradores, por exemplo, os microprogramas são idênticos, exceto no que se refere à micro-ordem que aciona a ULA para executar a operação lógica ou aritmética.

Para evitar a repetição exaustiva de tais micro-programas, que divergem apenas em um ponto, foi criada uma micro-ordem universal TIPS no campo FUNC que aciona a operação conveniente na ULA por decodificação do subcódigo de operação. Dessa maneira, foi possível fazer-se uma grande economia de espaço, pois para todas as instruções aritméticas e lógicas existe apenas um microprograma de execução, utilizando TIPS. (Fig. V.3)









microprograma de execução universal

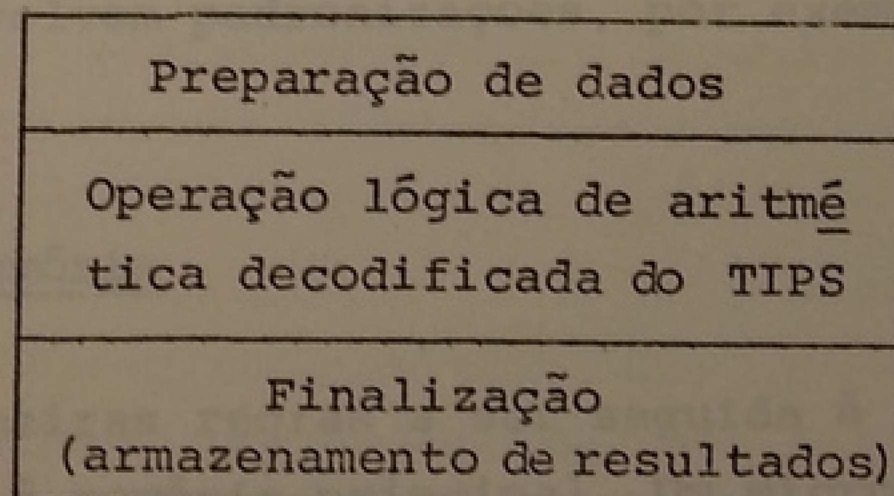
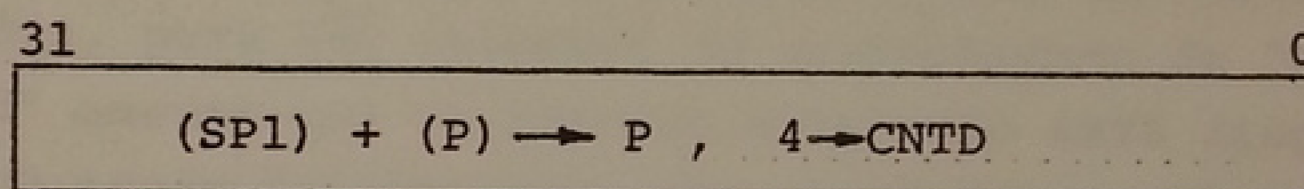


Figura V.3 - Microprogramas Comuns de Execução Utilizando Micro-ordem TIPS.

d) Paralelismo

Sob o ponto de vista microscópico, dentro de um microprograma, deve-se procurar aproveitar ao máximo o paralelismo de funções permitidas dentro de uma microinstrução. Esse paralelismo é determinado pela implementação de blocos funcionais em hardware mutuamente exclusivos, controláveis por micro-ordens também mutuamente exclusivas na microinstrução.

No G-10, por exemplo, pode-se efetuar uma operação na ULA, e ao mesmo tempo carregar uma constante no contador CNTD, na mesma microinstrução.



Esta propriedade, convenientemente utilizada, permite obter considerável economia de espaço e também de tempo.

V.2 OTIMIZAÇÃO EM TERMOS DE TEMPO

A otimização do tempo de execução de uma instrução é a mais importante, porém sujeita a considerações de espaço ocupado. De uma maneira geral, quanto mais rápida é uma micro-rotina mais especializada e particular é; assim, mais difícil se torna o uso de uma parte em comum com outra micro-rotina, pois chamadas de micro-subrotinas envolvem padronizações, por exemplo, dos registradores utilizados.

a) Acessos à memória

Uma das primeiras regras a ser seguida é a minimização do número de acessos à memória principal. Um microprograma eficien

te deve fazer acessos a registradores muito mais freqüentemente do que à memória principal que é comparativamente mais lenta [7]. Deve-se armazenar resultados intermediários em acumuladores ou registradores de trabalho. O acesso à memória principal só deve ser realizado para busca ou armazenamento de instruções ou dados do programa. Os microprogramas devem ser auto-suficientes quanto à geração ou utilização de constantes ou resultados parciais.

b) "Timing"

Os microprogramas devem levar em conta certos aspectos do "timing" da máquina: o ciclo de execução de operações no fluxo de dados e o ciclo de outras unidades funcionais devem ser compatibilizados, para não incorrer no aparecimento de "hazards". Um "hazard" assume que um dado ou resultado está disponível cedo demais (ou tarde demais) para uma execução correta.

Por exemplo, se uma microinstrução envolve um acesso para leitura de um dado na memória, e no ciclo de UCP seguinte a próxima microinstrução tentar utilizar o dado numa operação qualquer, o resultado da mesma pode estar incorreto, pois a memória não teve tempo suficiente para fornecer o dado desejado.

Uma das técnicas para resolver esse problema no G-10 foi a utilização de uma informação, vinda nas próprias microinstruções, sobre a disponibilidade ou não do dado.

Para atividades que requerem mais de um ciclo de controle para sua execução, pode-se aproveitar o tempo de espera do processador para efetuar operações que não dependam da memória. Como durante o acesso à mesma, o processador encontra-se inativo, é interessante percorrer a seqüência do programa para verificar se existe alguma operação com elementos do processador que não os registradores de dados (D), de endereços (E) e instruções (I), que eventualmente poderia ser efetuada durante esse intervalo. O número máximo de tais atividades pode ser calculado por:

$$N \leqslant \frac{T}{t}$$

onde T é o tempo de acesso ao dispositivo mais lento, e t é o ciclo de controle 7 .

c) Micro-rotinas críticas

Durante a otimização em tempo, deve-se dar maior atenção aos microprogramas freqüentemente usados, e sua otimização deve ser a melhor possível. Seqüências microprogramadas executadas repetitivamente ("loops") devem utilizar o máximo de paralelismo possível.

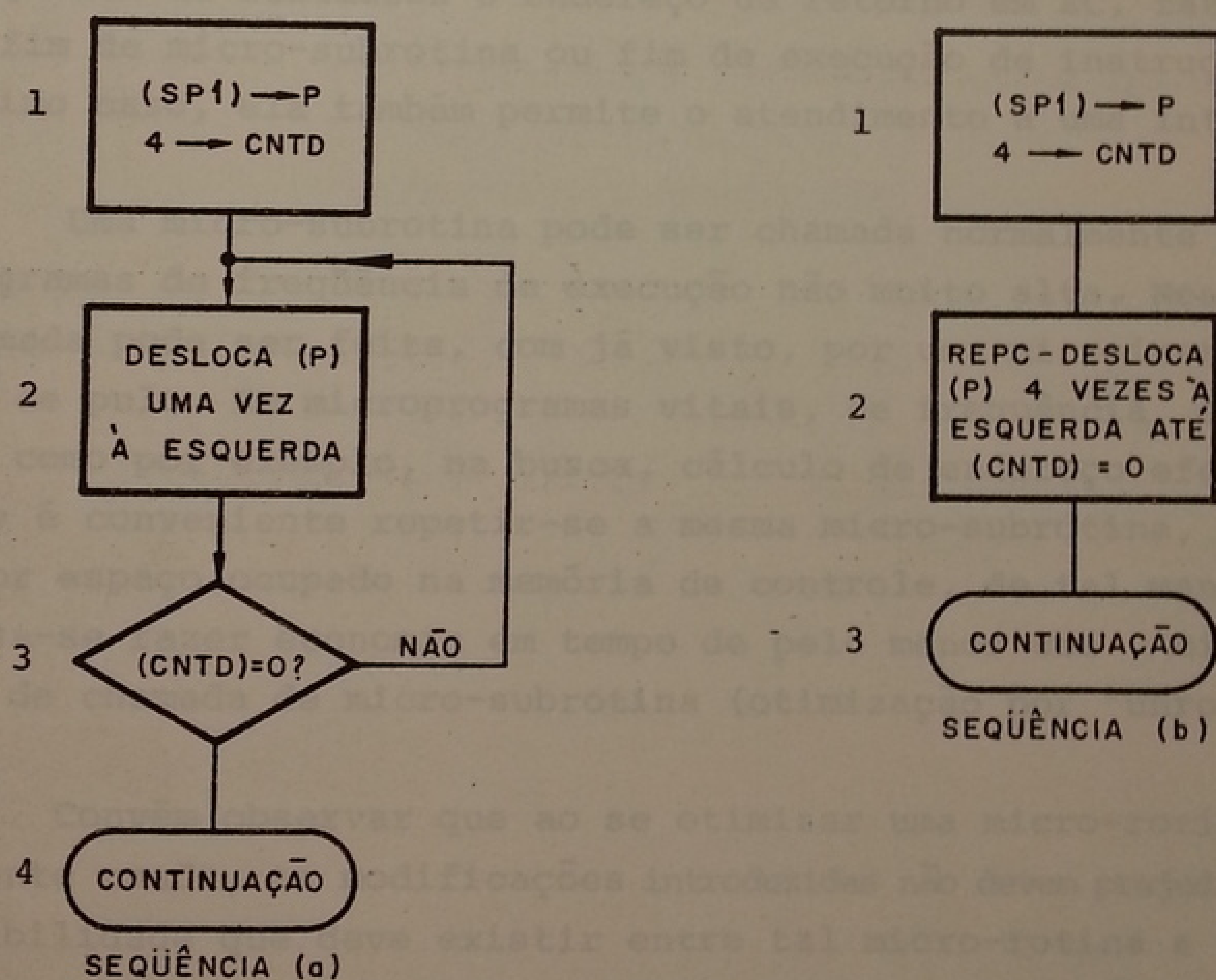


Figura V.4′ - Micro-ordem REPC

Na seqüência (a) o dado em P é deslocado, e a cada vez é testado o contador. Se o conteúdo deste não for nulo volta-se a um novo deslocamento. O "loop" consiste dos passos 2 e 3 (2 ciclos de controle).

Na sequência (b), o "loop" foi simplificado para apenas 1 passo (passo 2), com o uso da micro-ordem REPC que faz com que a microinstrução de deslocamento seja executada 4 vezes até o conteúdo do contador ser nulo.

Micro-ordens especiais como REPC são implementadas em hardware, depois de serem escritos e analisados vários microprogramas confirmando a necessidade de intervenção do hardware para automatizar certas operações.

Outro exemplo é o caso do PUGC (pulo para micro-subrotina) que, além de executar o desvio, guarda o endereço de retorno na pilha de registradores SO e S1, automaticamente. A micro - ordem FIM, além de restaurar o endereço de retorno em EC, faz o teste de fim de micro-subrotina ou fim de execução de instrução. Nesse último caso, ela também permite o atendimento a uma interrupção.

Uma micro-subrotina pode ser chamada normalmente por microprogramas de freqüência de execução não muito alta. Nesse caso a chamada pode ser feita, com já visto, por uma microinstrução geral de pulo. Em microprogramas vitais, de frequência elevadíssima, como por exemplo, na busca, cálculo de endereço efetivo e outras é conveniente repetir-se a mesma micro-subrotina, a custo de maior espaço ocupado na memória de controle, de tal maneira que possa-se fazer economia em tempo de pelo menos uma microinstrução de chamada de micro-subrotina (otimização por "unrolling").

Convém observar que ao se otimizar uma micro-rotina freqüentemente usada, as modificações introduzidas não devem prejudicar a compatibilidade que deve existir entre tal micro-rotina e todos os microprogramas que a chamam. Deve existir, portanto, um "acordo universal" entre esses microprogramas antes de ser introduzida uma modificação [7].

VI - CARACTERÍSTICAS IMPLEMENTADAS EM SOFTWARE/ HARDWARE/MICROPROGRAMAÇÃO COMPROMISSO E RESTRIÇÕES

O projetista de uma unidade central de processamento encontra-se atualmente num ambiente relativamente diferente do que no passado, quando os componentes para a implementação de um computador eram muito caros e não tão eficientes. O software da época era sobrecarregado com a responsabilidade de executar qualquer função mais complexa, sendo o hardware uma entidade estática, difícil de ser modificado e com pouca flexibilidade para expansão ou adaptação para uma aplicação diferente da qual foi destinado.

A microprogramação, como foi visto, veio a influir no projeto de uma UCP de tal maneira que o controle é centralizado, e os caminhos no fluxo de dados se tornam mais simples de serem controlados. Assim, a unidade de controle pode ser encarada como um pequeno processador dentro de uma UCP, constando de uma memória, registrador de endereço, registrador de dados, lógica de condições para decisões, e outros pequenos circuitos para a seqüencialização e geração de sinais.

Existindo, assim, o conceito de um processador interno controlando uma unidade central de processamento maior, surge o problema de decisão de que funções poderiam ser retiradas do software ou do hardware para serem transferidas à microprogramação. Isto porque o projetista se depara com a seguinte perspectiva: o constante aumento do uso de tecnologia MSI e LSI tende a baixar o custo do hardware, tornando-se o custo de desenvolvimento do software cada vez maior em relação ao hardware. O software, assim, pode ser aliviado de certos encargos que normalmente são repetitivos e ocupam espaço útil de memória; com a microprogramação, estas tarefas podem ser automatizadas com velocidades de execução ao nível de hardware, com economia de instruções e de memória para a sua implementação.

Um dos critérios de decisão para distribuição de funções entre hardware, software e microprogramação, é estabelecer um compromisso entre velocidade, freqüência de execução e complexidade.

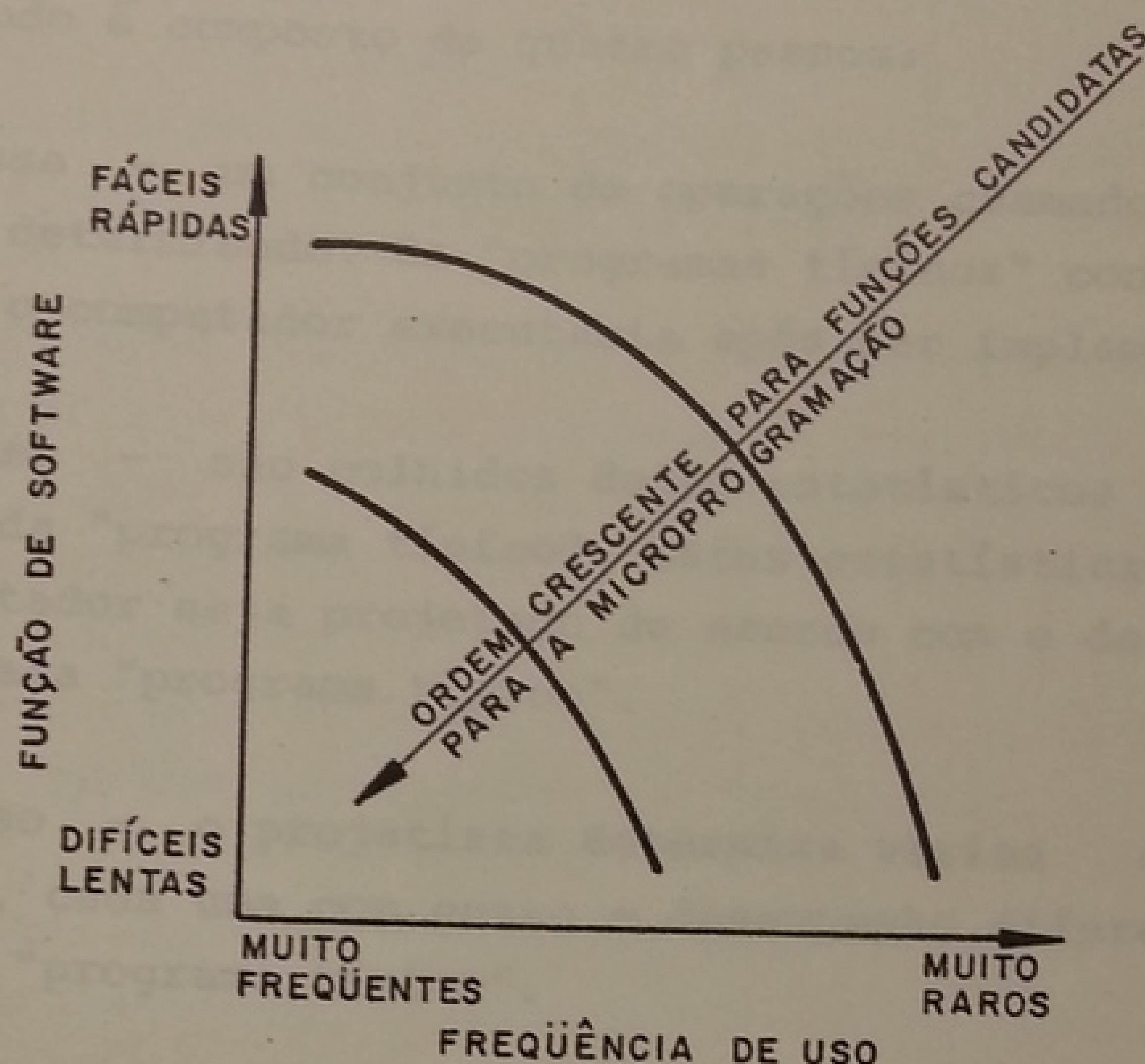


Figura VI.1 - Ordem de Avaliação para Funções Atribuídas à Microprogramação [9]

A figura VI.1 ilustra o processo de decisão. Funções de software que são freqüentemente usadas, porém difíceis de implementar ou lentas (à esquerda, próximas à origem das coordenadas) são fortes candidatas para a microprogramação. Como exemplo, pode-se citar o carregador-relocador microprogramado proposto por Chu [2], ou o carregador absoluto apresentado em detalhe mais adiante nesse capítulo. As funções que são relativamente fáceis de implementar e rápidas em software, e que ao mesmo tempo são raramente usadas, podem ainda, eventualmente, serem microprogramadas, tendo em vista, porém, os limites impostos pelo custo e eficiência e técnica de implementação.

Existem vários métodos para efetuar tal decisão. Um deles, desenvolvido por Hinshaw e Irani [4], consiste numa avaliação experimental, através da microprogramação, da eficiência e freqüência de utilização de certas operações-padrão, de acordo com

a qual será baseado o projeto da arquitetura de uma máquina.

O método é composto de quatro passos:

1º passo - um conjunto de operações chamado "programas típicos" é determinado. Os "programas típicos" constam das operações que o computador executaria após ser implementado.

2º passo - são colhidos dados estatísticos sobre a utilização de cada "programa típico". Estas estatísticas permitem que o computador seja projetado de acordo com o desempenho desejado para cada "programa típico".

3º passo - o projetista determina várias configurações em hardware, cada uma com custo e desempenho diferentes, de acordo com cada "programa típico".

4º passo - um programa de otimização seleciona e combina as várias implementações resultantes do 3º passo. O produto desse programa é a configuração de hardware de mínimo custo, que corresponde a um certo nível de desempenho desejado.

Baseando-se nas estatísticas obtidas sobre a utilização de "programas típicos", pode-se determinar qual a melhor maneira de implementar certas funções em hardware, microprogramação ou mesmo software. Tais "programas típicos", segundo Hinshaw e Irani podem ser operações aritméticas, operações de pilha, operações de listas ligadas, manipulação de cadeias de dados, operações de entrada e saída, desvios condicionais, etc.

De acordo com sua freqüência de utilização ou complexidade, pode-se determinar, para cada operação, qual a porcentagem de hardware, microprogramação ou software necessária para implementá-la.

De uma maneira geral, a atribuição de funções num computador se baseia, na prática, na experiência obtida em sistemas já em funcionamento, de tal maneira que as novas versões possam adquirir características mais otimizadas, com a simples alteração ou adição de novos microprogramas na memória de controle.

Em alguns casos, a microprogramação não tem outro efeito a não ser aumentar a velocidade por meio da eliminação dos acessos à memória quando da busca de instruções. Além disso, o microprogramador tem ao seu alcance todas as vias de dados, registradores e demais elementos lógicos do sistema, e principalmente o paralelismo inerente à arquitetura, características essas não acessíveis na sua totalidade pelo programador do software.

Os objetivos a serem alcançados pelo emprego da microprogramação na automatização de procedimentos são vários:

a) Desempenho melhor. A eficiência pode ser aumentada pela inclusão de características internas tais como: registradores indexadores acessíveis pelos microprogramas, externos à memória principal (em alguns sistemas esses registradores são posições fixas da memória), aritmética de ponto flutuante microprogramada, e até montadores, carregadores e compiladores microprogramados.

b) Minimização do custo do sistema. Como o custo do hardware vem diminuindo relativamente ao do software, este poderá ter algumas de suas funções transferidas para o hardware.

c) Redução da complexidade do software. Um dos objetivos que está se tornando aparente é a tendência a reduzir a complexidade de sistemas operacionais e programas de usuário, pela inclusão de características em hardware que reduzam o número de instruções, aumentem a eficiência em tempo de execução, e livrem o programador de problemas de limitação de espaço de memória. Exemplos que podem ser citados: gerenciamento automático da memória, proteção de áreas na memória, manipulação de seqüências de caracteres, compilação, operações de pilha, etc. [10].

d) Confiabilidade do sistema. O software é freqüentemente sujeito a falhas devidas a mudanças freqüentes e conseqüentes possíveis incompatibilidades entre programas novos e antigos. É conveniente, portanto, deslocar funções críticas para locais mais seguros, como por exemplo, para a microprogramação.

e) Para prolongar a vida de um sistema. No campo da computação, o tempo de vida de um sistema é, às vezes, medido pela possibilidade de sua mudança. Essa flexibilidade é conseguida transferindo-se funções implementadas em hardware para a microprogramação ou para o software. Nos sistemas microprogramados dedicados a comunicações, por exemplo, as mensagens enviadas podem ter formatos diferentes para cada tipo de comunicação, e microprogramas diferentes devem ser implementados para cada caso. Uma memória do tipo "ler-escrever" na unidade de controle facilita as mudanças desse gênero.

f) Compatibilização de sistemas diferentes. A implementação de emuladores é um exemplo de um tipo de flexibilidade que apenas os computadotes microprogramados possuem. No Capítulo VI são definidas a emulação è as técnicas de implementação de um emulador.

Durante o desenvolvimento do projeto G-10 procurou-se atribuir â microprogramação certas funções que se encontram no software de muitos minicomputadores antigos, e até de alguns bem modernos. Outras funções que normalmente são realizadas pelo hardware foram transferidas para a microprogramação no G-10, por uma questão de flexibilidade e facilidade de implementação. Estas funções são:

- Proteção à memória
- Busca da próxima instrução
- Endereçamento
- Operações de pilha

- Mudança de contexto em interrupções
- Carregador absoluto
- Emulação (em desenvolvimento).

As funções mais significativas são detalhadas a seguir.

a) Proteção de memória

O G-10 oferece a facilidade de implementação de um sistema multiprogramado, onde cada usuário possui uma área de programas e uma área de dados, sendo, qualquer posição de memória fora dessa área, protegida. Essa proteção consiste em testar se cada instrução ou dado do programa desse usuário está dentro dos limites de sua área. Esses limites são demarcados por dois registradores-base (base de programa e base de dados) e dois registradores-limite (limite de programa e limite de dados).

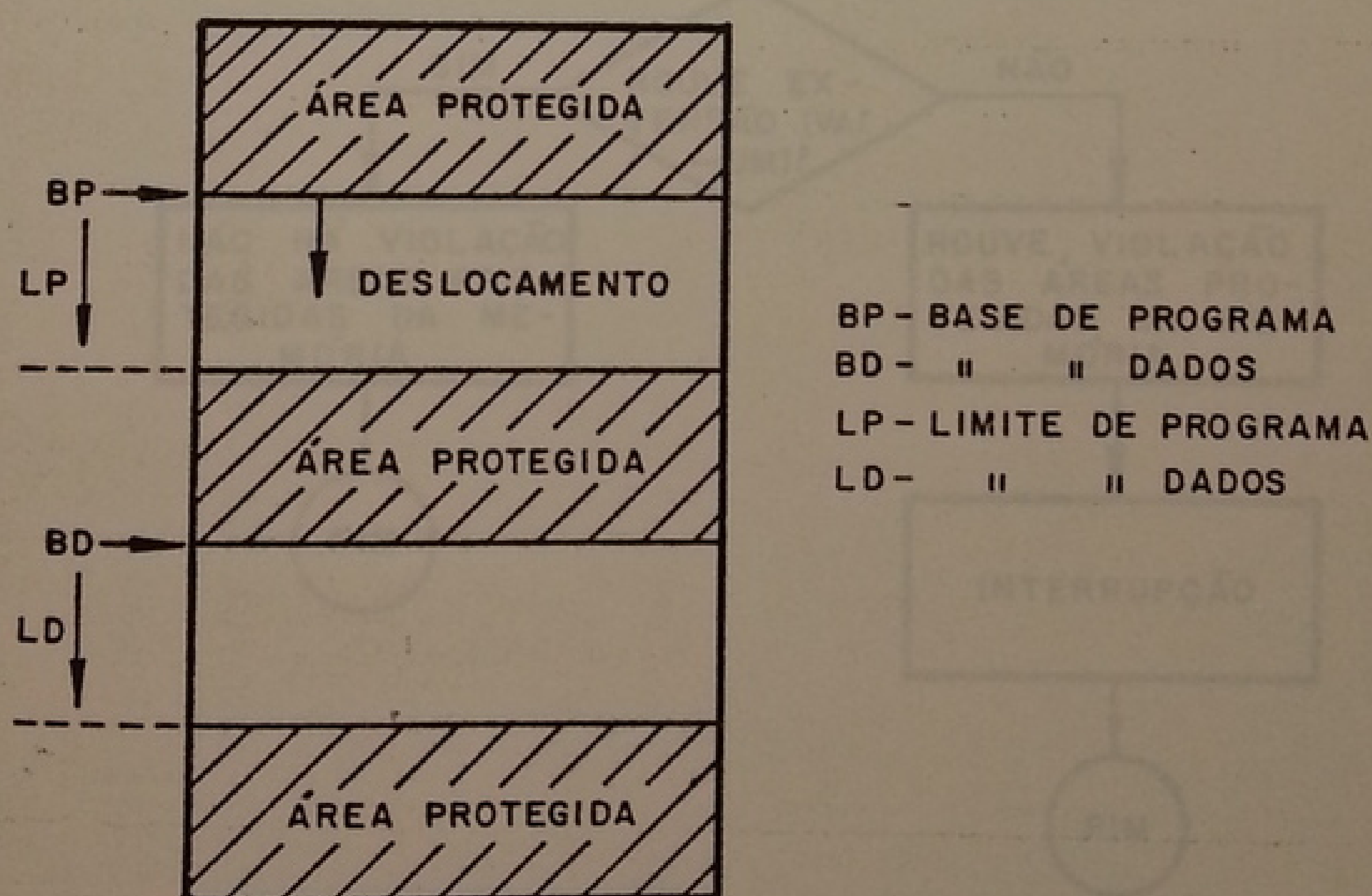


Figura VI.2 - Divisão Lógica da Memória

As bases são valores absolutos de endereços na memória, e os limites são valores relativos às bases. Por exemplo, se a área de programas contém 20 posições, a base de programa poderia ser 100, e o limite de programas seria 20. Se o programador deseja acessar uma palavra na posição 11 do seu programa, ele estará acessando, na verdade o endereço 111 (base de programa + deslocamento em relação a esta base) que é o endereço absoluto dessa palavra na memória.

O teste de proteção consiste em verificar se o deslocamento em relação à base é menor do que o limite dessa área.

Em termos de microprogramação, isso corresponde ao seguinte diagrama em blocos:

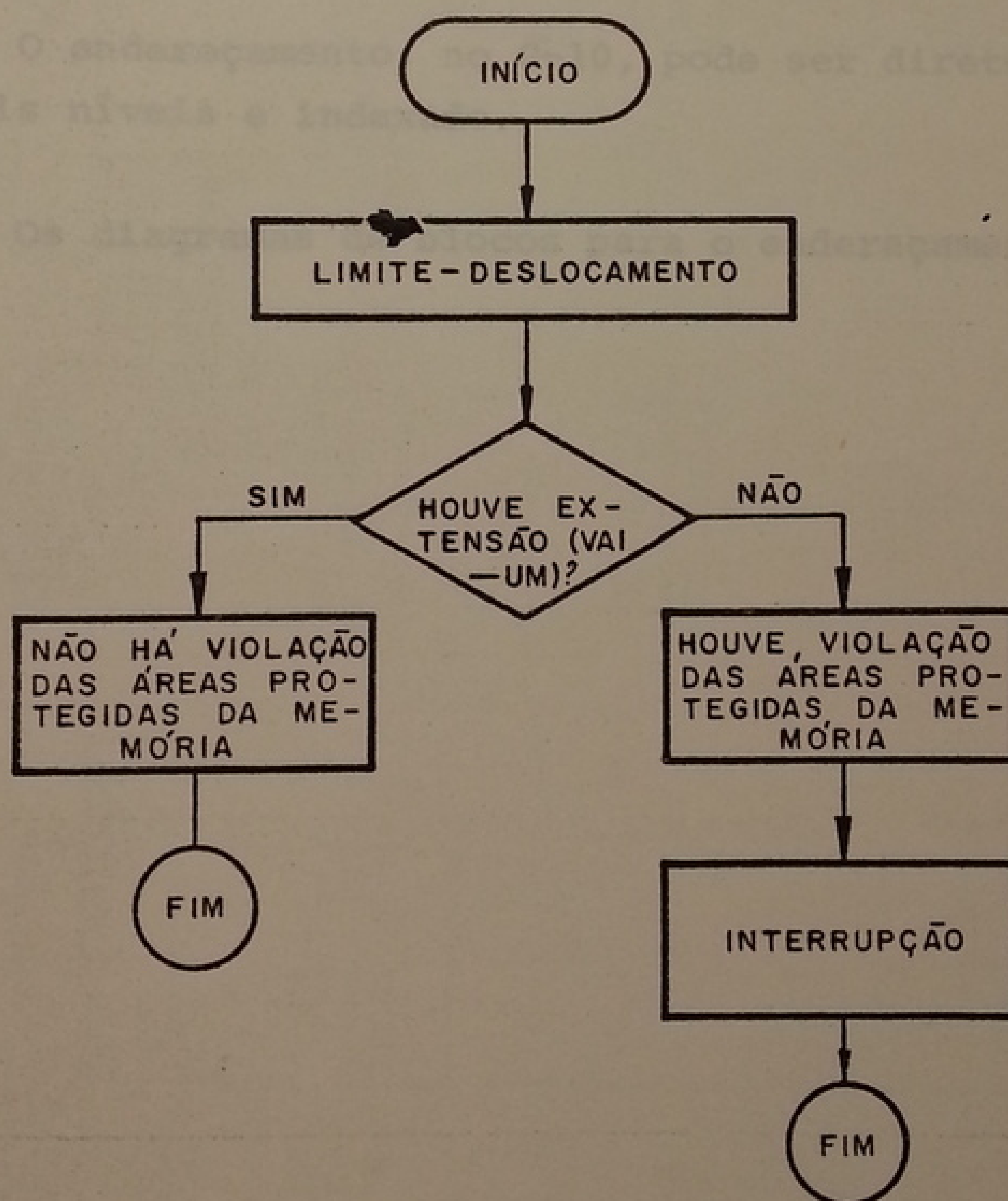


Figura VI.3 - Proteção de Memória

Este diagrama representa uma micro-subrotina que é chamada a cada acesso para leitura da memória. Caso a operação seja uma escrita, o teste de proteção deve ser efetuado antes da mesma, para não destruir a área de outro usuário.

Esta micro-rotina é padrão para todos os microprogramas que tem um acesso à memória. O software, nesse caso, terá apenas a função de gerenciar as áreas, no sentido de alocá-las na memória para cada usuário, sem a necessidade de recorrer a testes constantes de proteção durante a execução de cada instrução do programa de um determinado usuário.

b) Endereçamento

O endereçamento, no G-10, pode ser direto, indireto de um ou dois níveis e indexado.

Os diagramas de blocos para o endereçamento são os seguintes:

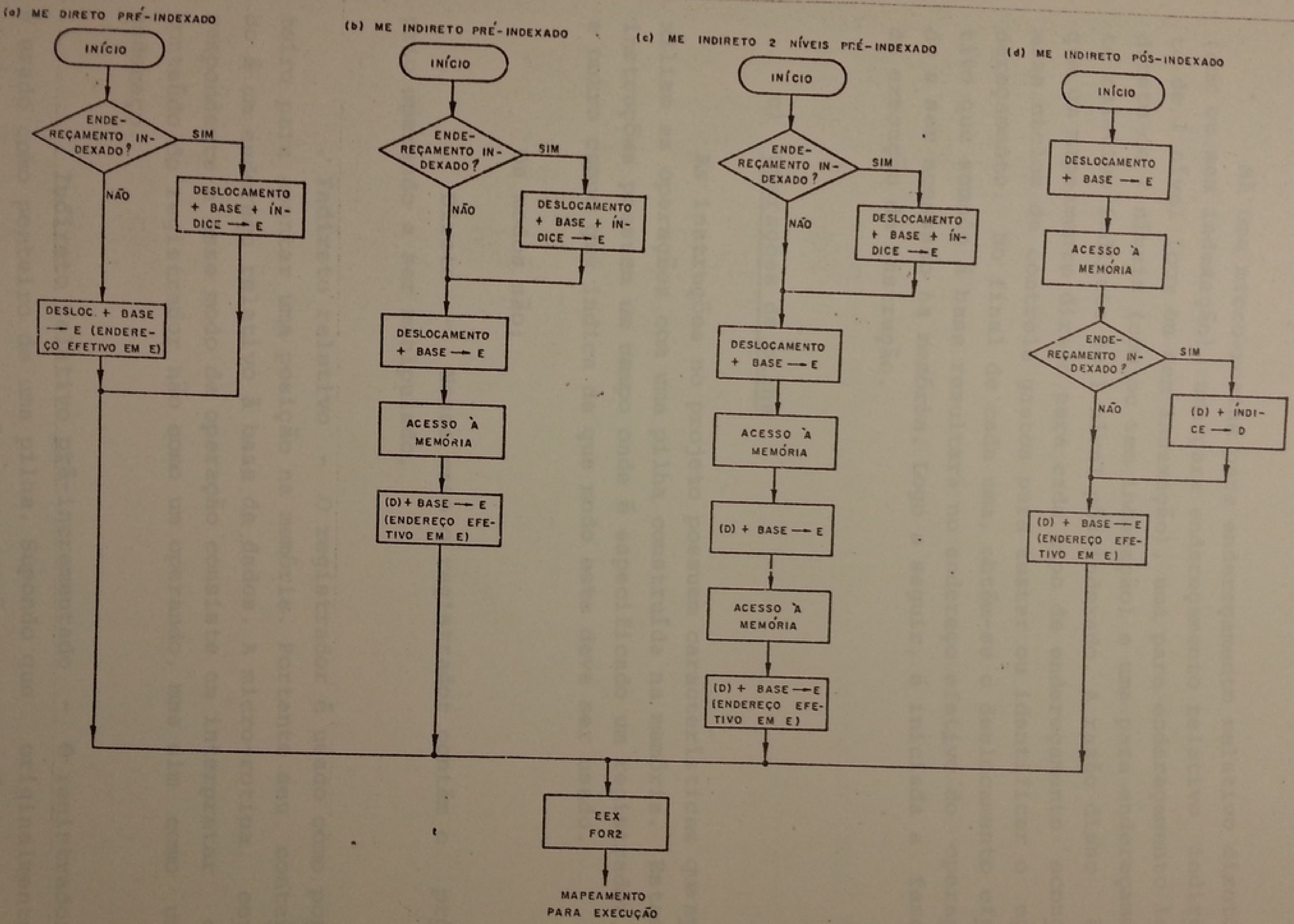


Figura VI.4 - Endereçamento

Há uma micro-rotina para endereçamento relativo direto (com ou sem indexação), uma para endereçamento relativo indireto de 1 nível (com ou sem indexação), uma para endereçamento indireto de 2 níveis (com ou sem indexação) e uma para endereçamento relativo indireto de 1 nível pós-indexado. A razão disso é que um mapeamento direto para cada tipo de endereçamento economiza ciclos de controle gastos para testar ou identificar o endereçamento. No final de cada uma, obtém-se o deslocamento efetivo que somado à base resultará no endereço efetivo do operando a ser acessado na memória. Logo a seguir, é iniciada a fase de execução da instrução.

c) Operações de pilha

As instruções no projeto possuem características que auxiliam as operações com uma pilha construída na memória. Estas instruções possuem um campo onde é especificado um registrador, e outro campo que indica de que modo este deve ser usado.

Os modos são:

. Direto - Nesse caso o registrador contém o próprio operando a ser manipulado.

. Indireto relativo - O registrador é usado como ponteiro para apontar uma posição na memória. Portanto seu conteúdo é um endereço relativo à base de dados. A micro-rotina correspondente a esse modo de operação consiste em interpretar o conteúdo do registrador não como um operando, mas sim como um endereço.

. Indireto relativo pré-incrementado - O registrador é usado como ponteiro de uma pilha. Supondo que originalmente ele esteja apontando para a última posição ocupada, é previamente incrementado para apontar a próxima posição onde será armazenado o dado.

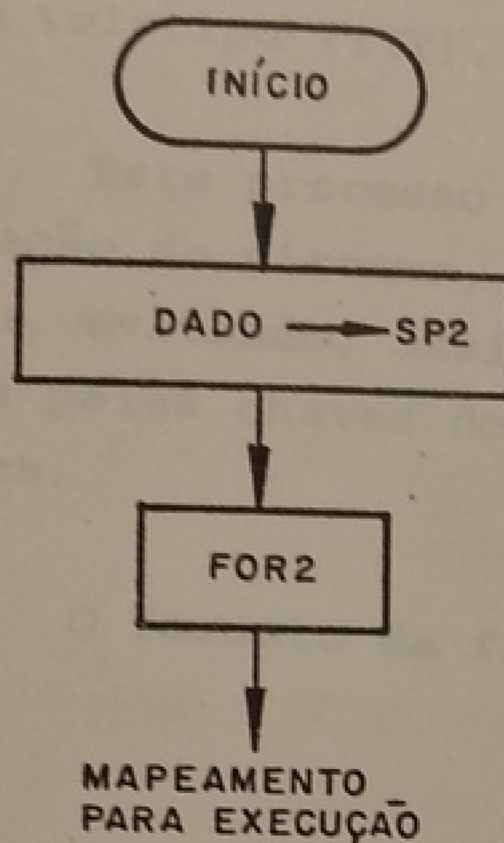
. Indireto relativo pós-decrementado - Neste caso, o registrador é usado como ponteiro apontando a última posição ocupada na pilha. Numa operação de retirada de dados, este é decrementado de uma posição após o acesso ao dado na memória.

Os dois últimos modos de operação são os de uma pilha, correspondentes a "PUSH" e "POP".

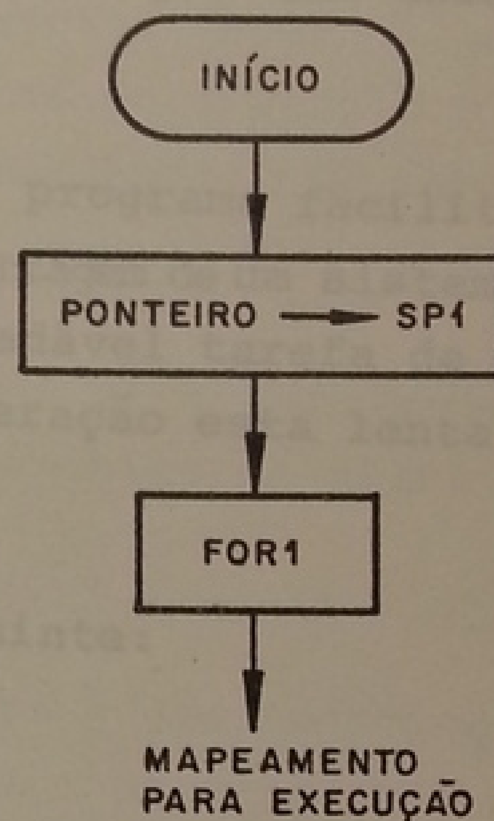
Para cada um dos quatro modos existe uma micro-rotina que é acessada após a fase de busca da instrução.

Os diagramas de blocos seguintes ilustram os vários modos de operação:

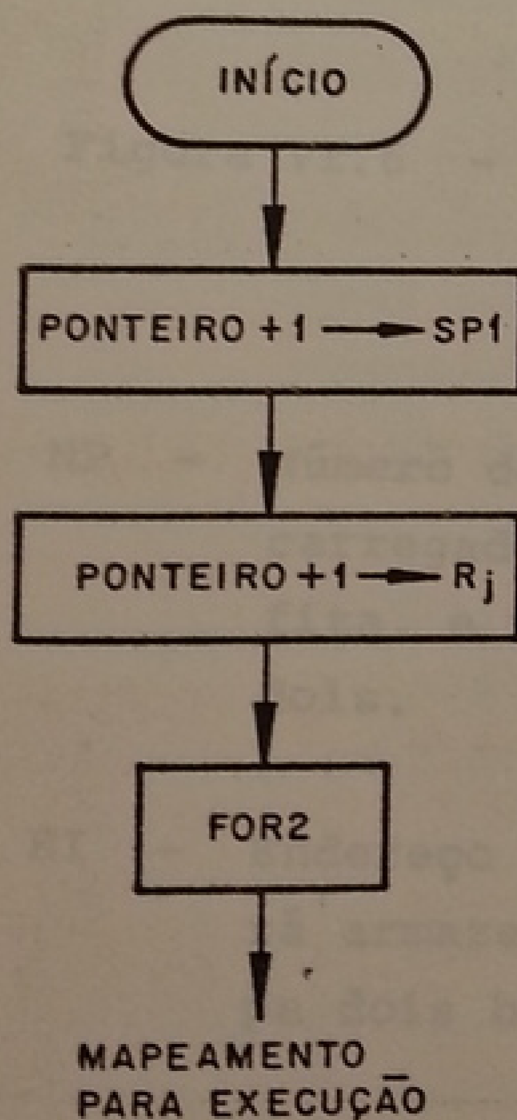
a) MO DIRETO



b) MO INDIRETO



c) MO INDIRETO PRÉ-INCREMENTADO



d) MO INDIRETO PÓS-DECREMENTADO

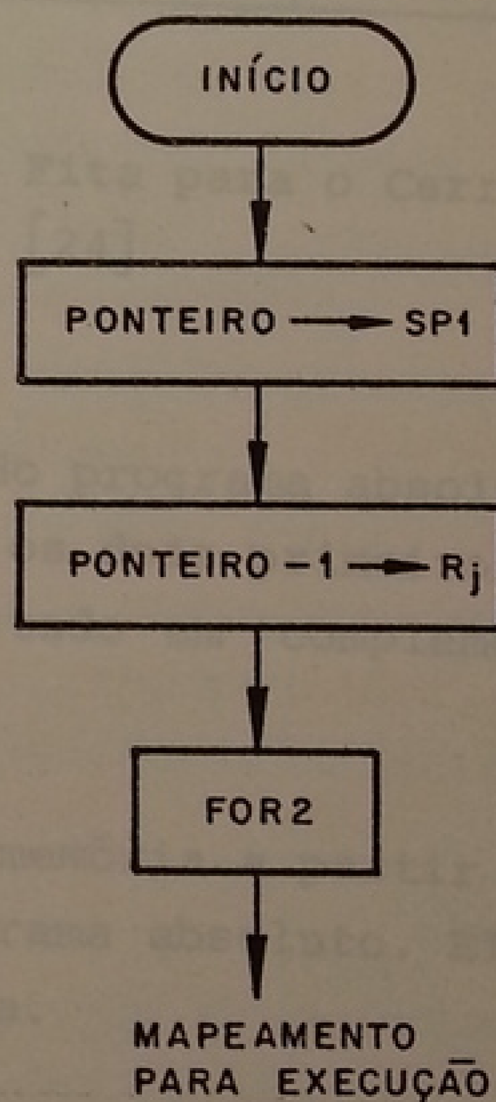


Figura VI.5 - Operações de pilha

d) Carregador absoluto

É um microprograma que carrega um programa com endereçamento absoluto a partir de um endereço na memória, mediante o acionamento de um botão no painel da máquina; o programa deve estar perfurado numa fita de papel, lida por uma unidade rápida ou por um teletipo (TTY).

Este processo de carga de programa facilita muito a inicialização do sistema, quando da montagem de um sistema monitor na memória, evitando, assim, a desagradável tarefa de carga de instruções pelas chaves do painel, operação esta lenta e passível de erros.

O formato da fita é o seguinte:

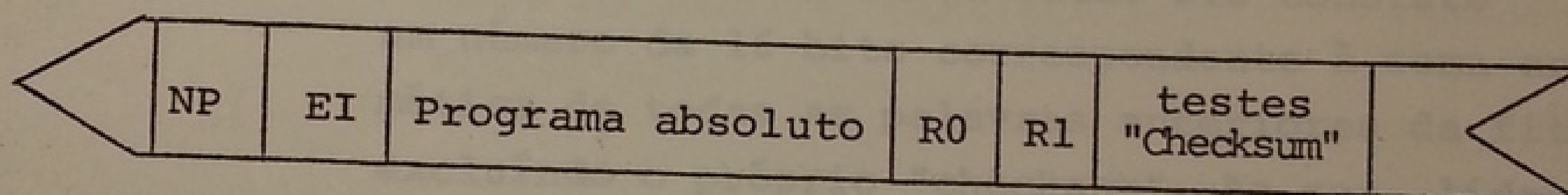


Figura VI.6 - Formato da Fita para o Carregador Absoluto [24]

NP - Número de palavras do programa absoluto a ser carregado. NP ocupa os dois primeiros bytes da fita, e vem representado em complemento de dois.

EI - Endereço inicial na memória a partir do qual será armazenado o programa absoluto. EI também ocupa dois bytes da fita.

Programa absoluto - É o programa a ser carregado. Cada palavra é representada por dois bytes na fita, e o byte mais significativo é lido antes do menos significativo.

R0 - É a posição na memória a partir da qual será executado o programa, ou parte do programa. Por exemplo, quando o programa absoluto contém constantes nas suas primeiras posições o endereço da primeira instrução a ser executada é diferente do endereço inicial especificado por EI; R0 é a posição inicial de execução a ser carregada no contador de instruções. R0 ocupa dois bytes da fita.

R1 - É uma constante que será carregada no registrador R1, que é o ponteiro da pilha. R1 ocupa dois bytes da fita.

Teste ("Checksum") - Esta é uma informação que ocupa os dois bytes finais da fita. Ela consiste de um número de 16 bits correspondente à soma aritmética de todas as palavras anteriores da fita, excluindo o próprio "checksum". Ao ser lida cada palavra a partir da fita, seu conteúdo é somado ao resultado parcial da soma resultante das palavras anteriores. O resultado final dessas somas em série é comparado ao "checksum" que consta na fita. Se eles forem iguais, então a fita foi lida corretamente, e as informações na memória são válidas; caso contrário alguma informação não foi lida corretamente, o que será avisado pelo painel do computador, onde todos os bits dos sinais luminadores sãofeitos iguais a 1.

O carregador absoluto é ativado pressionando-se o botão CAR no painel quando o processador estiver no estado parado.

O diagrama de blocos do microprograma do carregador absoluto vem descrito na figura VI.7.

Os parâmetros do programa estão assim localizados:

Registrador R.6 - contém o resultado das somas das palavras lidas a partir da fita.

Registrador R.7 - contador contendo o número de palavras do programa absoluto. Ele é decrementado após cada leitura de dois bytes.

Registrador R.B - contém a instrução Testa Estado do dispositivo para a leitora de fita.

Registrador R.C - contém a máscara da instrução Testa Estado.

Registrador R.D - contém a instrução Entra Dado, para a leitora do dado da interface para o processador central.

INICIO — ENDEREÇO INICIAL: NUPAL

INICIALIZAÇÃO DOS REGISTRADORES COM CONSTANTES E INSTRUÇÕES PARA O CANAL E DISPOSITIVO, QUE SERÃO USADOS NO DECORRER DA EXECUÇÃO DO CARREGADOR ABSOLUTO.

/0000 → R.6

INSTRUÇÃO TESTA ESTADO → R.B

MÁSCARA DA INSTRUÇÃO ALTERA ESTADO → R.C

INSTRUÇÃO ENTRA DADO → R.D

LEITURA DE UM BYTE DA FITA — MICRO-ROTINA INIC

INSTRUÇÃO ALTERA ESTADO → REGISTRADORES I, E

MÁSCARA ALTERA ESTADO → REGISTRADOR D

REQUISIÇÃO DA VIA COMUM: FOI DADO O COMANDO PARA LEITURA DE BYTE DA FITA

TESTE DE "OCUPADO" DO DISPOSITIVO. SE JÁ ESTIVER "LIVRE", LÊ BYTE DA FITA. — MICRO-ROTINA LOOPD

INSTRUÇÃO TESTA ESTADO → REGISTRADORES I e E

MÁSCARA DE TESTA ESTADO → REGISTRADOR D

REQUISIÇÃO DA VIA COMUM: A PALAVRA DE ESTADO DO DISPOSITIVO É LIDA

(1)

LOOPD

(1)

O DISPOSITIVO AINDA ESTA OCUPADO? — SIM → REEXECUÇÃO DA MICRO-ROTINA LOOPD

NÃO ↓

INSTRUÇÃO ENTRA DADO → REGISTRADORES I e E

REQUISIÇÃO DA VIA COMUM: É LIDO UM BYTE DA INTERFACE PARA O PROCESSADOR

TESTA SE O 1º BYTE DA FITA FOI LIDO. A PALAVRA NP É LIDA E CARREGADO. — MICRO-ROTINA NUP1

BYTE LIDO É ZERO? — SIM → REEXECUÇÃO DA MICRO-ROTINA INIC

NÃO ↓

PULO PARA MICRO-SUBROTINA BYTE1 QUE LÊ O BYTE MENOS SIGNIFICATIVO DA PALAVRA E COMPACTA OS 2 BYTES

PALAVRA LIDA + CONTEÚDO DE R.6 (CHECK SUM) → R.6

PALAVRA LIDA (QUE É O NÚMERO DE PALAVRAS NP DO PROGRAMA ABSOLUTO) → R.7

(2)

(2)

MICRO-ROTINAS GERAL E BYTE1

EXECUÇÃO DAS MICRO-ROTINAS INIC E LOOPD PARA LER BYTE SEGUINTE (BYTE MAIS SIGNIFICATIVO)

DESLOCAMENTO DO BYTE LIDO DE 8 POSIÇÕES À ESQUERDA

EXECUÇÃO DE INIC E LOOPD PARA LER BYTE MENOS SIGNIFICATIVO

BYTE MAIS SIGNIFICATIVO + BYTE MENOS SIGNIFICATIVO → PALAVRA COMPACTADA.

MICRO-ROTINA INMEM: LÊ O ENDEREÇO INICIAL DE CARGA NA MEMÓRIA.

PALAVRA LIDA + CONTEÚDO DE R.6 → R.6

PALAVRA LIDA (QUE O ENDEREÇO INICIAL EI DE CARGA DO PROGRAMA ABSOLUTO → REG. A

"LOOP" DE LEITURA DE UMA PALAVRA DA FITA E TRANSFERENCIA PARA A MEMÓRIA. — MICRO-ROTINA LOOPC

EXECUÇÃO DAS MICRO-ROTINAS GERAL E BYTE1 P/LER UMA PALAVRA

PALAVRA LIDA + CONTEÚDO DE R.6 → R.6

CONTEUDO DE 0 → REGISTRADOR E

(3)

Figura VI.7.a - Microprograma do Carredor Absoluto

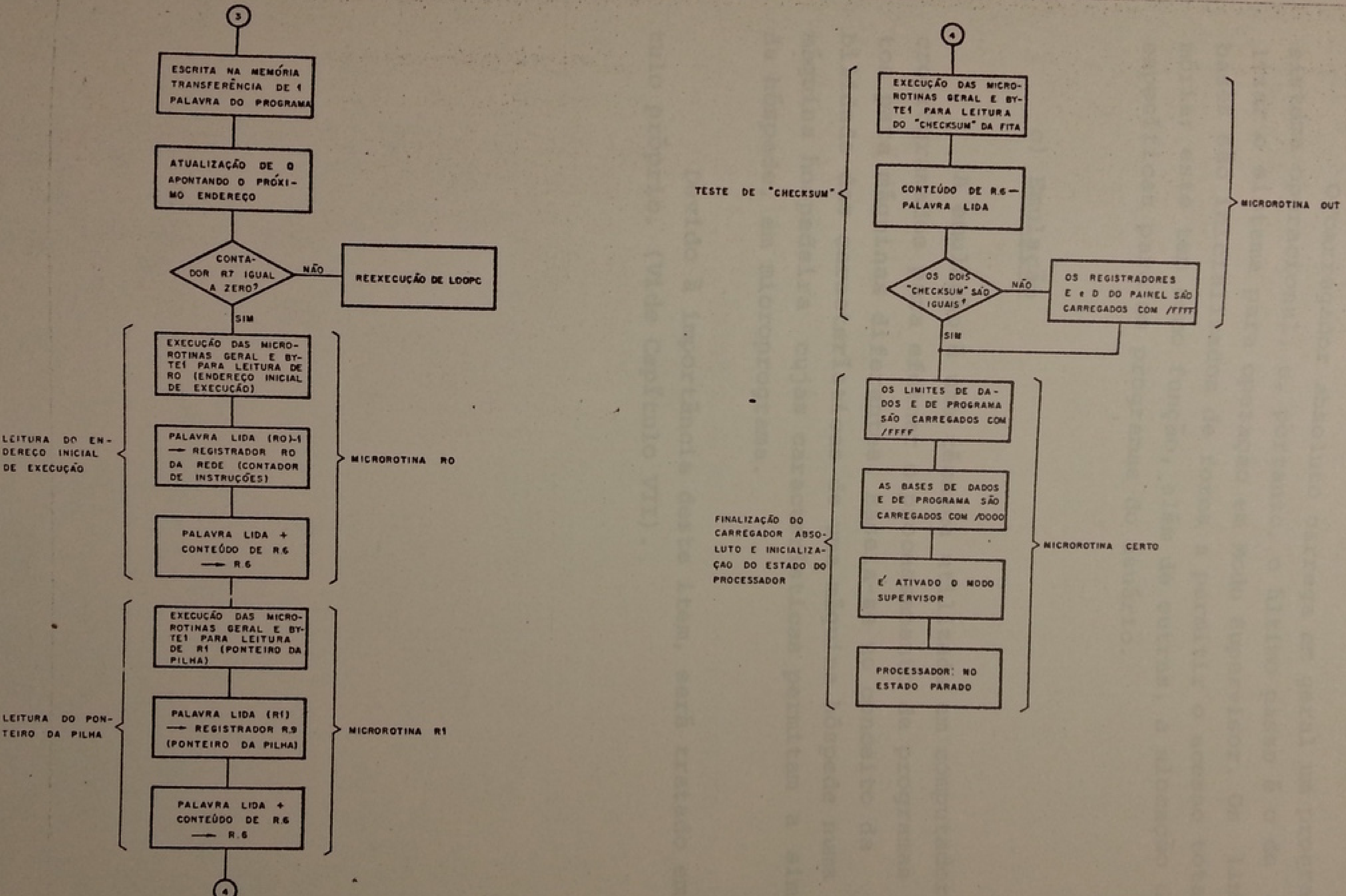


Figura VI.7.b - Microprograma do Carregador Absoluto

O carregador absoluto carrega em geral um programa de sistema operacional, e, portanto, o último passo é o de inicializar o sistema para operação em Modo Supervisor. Os limites e bases são inicializados de forma a permitir o acesso total à memória; este tem como função, além de outras, a alocação de áreas específicas para os programas do usuário.

e) Emulação

A emulação é uma técnica utilizada em computadores microprogramados para efetuar o processamento de programas escritos para máquinas diferentes. Baseia-se no conceito da adaptabilidade das características de uma máquina hóspede numa outra máquina hospedeira cujas características permitam a simulação da hóspede, em microprograma.

Devido à importância deste item, será tratado em capítulo próprio. (Vide Capítulo VII).

VII - EXEMPLO DE APLICAÇÃO DE MICROPROGRAMAÇÃO: EMULAÇÃO

BIBLIOTECA
Dept.º Eng. ELETRICIDADE
E. P. U. S. P.

VII.1 INTRODUÇÃO

Durante longo tempo, a maneira mais utilizada para o processamento de programas de uma máquina em outra, pela falta da primeira, tem sido a simulação. Esta técnica encontra aplicação freqüente na depuração de programas escritos para um computador que por alguma razão não esteja disponível, ou em pesquisas para a seleção de um conjunto de instruções para um computador ainda em fase de projeto.

A simulação resulta em geral num desempenho um tanto precário quanto ao tempo de processamento, quando comparado com o tempo na máquina original.

Com o advento de computadores microprogramados, surgiu uma nova técnica de simulação, chamada emulação, que, segundo Husson, [5], diminui o tempo de simulação de um fator 1:10. A emulação consiste basicamente numa simulação a nível de microprogramação. Isso tornou-se praticável com unidades de controle microprogramadas, sendo possível implementar as novas instruções com uma alteração básica do conteúdo da memória de controle, e alguns aspectos do hardware.

A emulação mostra-se particularmente útil quando os fabricantes de computadores têm se defrontado com o problema de compatibilização entre sistemas atuais com sistemas antigos.

Exemplo desse fato, é a emulação do sistema IBM 7080 no sistema IBM 360/65 [5] e do emulador do sistema Burroughs 220 no sistema IBM 360/25 [7].

A seguir serão ilustradas algumas técnicas de conversão de programas de um sistema para outro, conceitos fundamentais de emulação, e os problemas envolvidos na compatibilização de sistemas. Antes, porém, justifica-se a definição de alguns termos que serão empregados nesse capítulo.

MÁQUINA HÓSPEDE: é a máquina cujas características se deseja representar em outro sistema, a máquina hospedeira, e cujos estados possam ser interpretados pelos elementos e recursos deste sistema. Enfim, é o computador cujos programas se quer processar em outra máquina.

MÁQUINA HOSPEDEIRA: é o sistema cujos elementos e características são capazes de representar os elementos da hóspede através de registradores, memória principal e outros recursos.

IMAGEM: é a representação da hóspede na hospedeira, isto é, existe um elemento na hospedeira que, eventualmente em conjunto com outras facilidades, é capaz de exercer a mesma função do elemento correspondente da hóspede. Por exemplo, um acumulador de 16 bits do fluxo de dados do sistema IBM 1130 pode ser representado por um registrador de propósito geral do G-10.

VII.2 ESTUDO COMPARATIVO DAS VÁRIAS TÉCNICAS PARA A IMPLANTAÇÃO DE PROGRAMAS DE UMA MÁQUINA EM OUTRA

Há várias maneiras de abordar o problema de conversão de um programa escrito para uma máquina para ser processado em outra. Estes métodos podem envolver desde uma reprogramação da rotina, tradução de códigos, simulação, até a emulação. Estas técnicas diferem quanto ao esforço gasto, custo, eficiência com que a hospedeira é usada, e quanto às modificações necessárias na mesma para reconhecer e executar programas da hóspede.

Estas técnicas podem ser representadas pela seguinte escala:

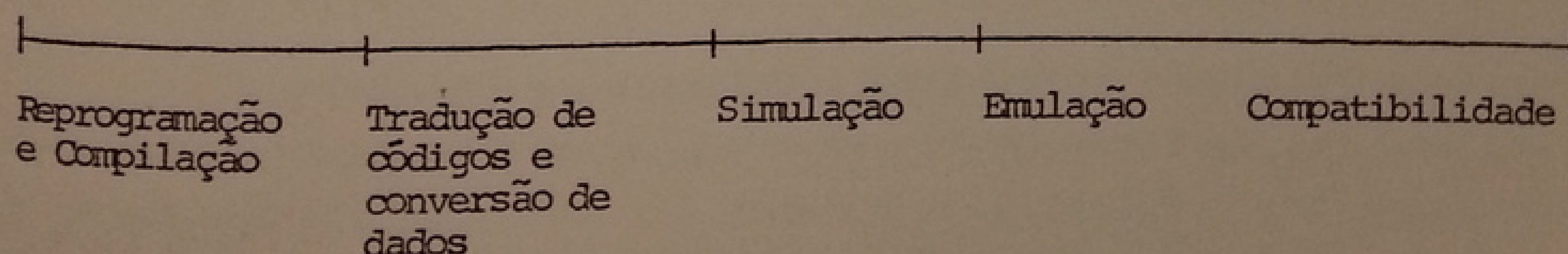


Figura VII.1 - Técnicas de Conversão [5]

a) Reprogramação

No extremo esquerdo da escala, o programador deve refazer a codificação do problema com o conjunto de instruções da máquina hospedeira. Isto seria vantajoso se ele quiser aproveitar as características e facilidades da hospedeira na sua totalidade. O custo, porém, é elevado, pois o método é muito trabalhoso e demorado no sentido de depurar o programa e o tempo de processamento gasto pode ser alto. Se o programa estiver codificado em uma linguagem de alto nível (COBOL, FORTRAN, etc.), seria necessário recompilá-lo para o sistema hospedeiro. [5] . Entretanto, diferenças na implementação dos compiladores costumam levar à necessidade de uma reprogramação maior ou menor.

b) Tradução de códigos

A tradução de códigos é um dos métodos menos eficientes. Cada instrução de máquina no programa fonte da hóspede é substituída por uma ou mais instruções que executam a mesma operação das instruções da hóspede. O programa fonte assim gerado para a hospedeira será montado e executado. Os dois fatores que influem na eficiência e desempenho na hospedeira são: as semelhanças de arquitetura entre os dois sistemas e a freqüência com que o código objeto é executado.

Um dos problemas da tradução é que para certas instruções da hóspede é preciso escrever virtualmente um programa em código da hospedeira, e quanto maior for a diferença entre os sistemas, mais complexo será o programa.

No caso de um programa automodificável, este método não pode ser aplicado.

c) Simulação

A simulação é definida como sendo o processo de representação do funcionamento de um sistema em outro. É um processo de modelagem do comportamento físico de uma máquina em outra, isto é, a imagem da máquina simulada é mapeada, bit a bit, na hospedeira.

Um simulador é constituído de blocos funcionais bem definidos que interagem entre si, em software. A parte mais importante é a rotina de interpretação que efetua a busca da instrução da hóspede, conversão de endereços para a máquina hospedeira, manipulação dos códigos de operação para obter os endereços das rotinas da hospedeira que executam as operações da instrução.

As rotinas de interfaceamento com a entrada e saída constituem um trabalho complicado. Quanto maior a diferença entre o funcionamento dos dispositivos de entrada e saída e características de dados da hóspede e hospedeira, maior é a complexidade dessa parte.

A vantagem da simulação em software é que este método não requer conhecimento do hardware, baseando-se em apenas um modo de operação interpretativo. As desvantagens são a velocidade baixa de sua execução, e a inabilidade de um programa simulador conservar o paralelismo que existe na máquina hóspede, e interpretá-lo convenientemente na hospedeira.

d) Emulação

O ponto mais próximo à compatibilidade na escala de conversões de programas é a emulação.

As definições de emulação geralmente são divididas em duas categorias: as funcionais e as procedimentais. As definições funcionais se referem à meneira pela qual é implementado

um emulador, enquanto que as procedimentais se referem apenas ao comportamento externo de um procedimento emulado. [16]

VII.3 DEFINIÇÕES FUNCIONAIS E PROCEDIMENTAIS

a) Definição Funcional

A definição original de emulação se aplica a um método que combina a implementação de características em hardware e software que simulem um programa. O emulador era basicamente um simulador em software, enriquecido por certas características especiais em hardware (para maior compatibilidade) a fim de aumentar a velocidade do simulador. A implementação de características em hardware era efetuada apenas nos pontos críticos onde o impacto causado pelo aumento da velocidade fosse muito grande, esses pontos sendo determinados pela análise do funcionamento do simulador. A influência do hardware, nesses casos, recaía sobre a implementação de características que auxiliassem a execução de um pequeno conjunto de instruções novas na máquina hospedeira para melhorar o processo da emulação. A razão principal para limitar o número de modificações em hardware era o custo; no caso das máquinas microprogramadas, a memória de controle era um item muito caro e portanto, um recurso limitado.

Mais recentemente, uma nova definição foi aceita, no sentido de aproximar mais o conceito de emulação a uma compatibilidade total de hardware; isto é conseguindo através da implementação microprogramada de todo o conjunto de instruções da máquina hóspede. O hardware, nesse caso, poderia auxiliar apenas no suporte de operações de entrada e saída.

b) Definição Procedimental

Aqueles que preferem a definição procedimental de emulação, encaram-na apenas sob o ponto de vista de que o emulador é uma caixa preta, e o que importa são as entradas e saídas,

sem levar em conta o que realmente ocorra, contanto que se produzam saídas (ou efeitos) realmente compatíveis com o funcionamento da hóspede. Assim, uma réplica passo a passo da hóspede na hospedeira não é necessária. Apenas os resultados finais devem ser idênticos.

De uma maneira bem geral, dentro da definição procedimental, uma vez definida a máquina hóspede, qualquer implementação do funcionamento da mesma na hospedeira é uma emulação. Esta noção se aplica principalmente a famílias de sistemas. Por exemplo, todos os modelos IBM/360 são emuladores do sistema /360 ; cada modelo é uma implementação diferente, acarretando resultados diferentes apenas no desempenho.

Tanto a definição funcional como a procedimental da emulação excluem a dependência com o tempo de execução de programas da máquina a ser emulada. Isto é, o programa processado na hóspede não é necessariamente processado com a mesma duração quando emulado na hospedeira. Muitas vezes, principalmente quando a hóspede é uma máquina de geração anterior à hospedeira e portanto mais lenta, seus programas são processados mais rapidamente nesta. Como exemplo, pode-se citar a diferença dos ciclos de acesso à memória principal entre o IBM/1130 (3,2us) e o do G-10 (0,85 us).

Para o estudo de emulação neste trabalho, cabe optar por uma das definições acima expostas, sem querer restringir o o conceito para uma determinada definição, mas sim, para focalizar um aspecto de emulação, cuja implementação seja mais direta, sem incorrer em transformações muito grandes na máquina hospedeira. Será abordada, portanto, apenas a técnica de implementação das instruções da hóspede na hospedeira com o mínimo de alterações em hardware possíveis.

Será adotada, assim, a definição procedimental, assumindo que o emulador seja uma caixa preta, cuja entrada seria o programa objeto da hospedeira, e cuja saída seria a execução das instruções com o hardware da hóspede, sendo o estado final da hospedeira um conjunto de informações constituintes do estado final da hóspede, após a execução de cada instrução.

A emulação, de agora em diante, será referida como "uma execução interpretativa das instruções da hóspede ao nível de linguagem de máquina por uma máquina hospedeira microprogramada ou microprogramável". [2]

VII.4 MÁQUINA HOSPEDEIRA E MÁQUINA HÓSPEDE

Um sistema emulador requer, para seu funcionamento que exista uma imagem completa da máquina hóspede na hospedeira. Diferentemente do simulador, que, por definição, é implementado apenas em software e é processado basicamente na arquitetura da hospedeira, o emulador pode utilizar registradores internos da hospedeira e outras características de hardware, como a lógica de testes de condições, contadores, canais de entrada e saída, etc., que representam o estado verdadeiro do microprograma emulador.

Supondo serem definidas duas máquinas, S_1 e S_2, com uma correspondência biunívoca entre seus estados; após a execução de uma operação, para cada estado S^i_1 na máquina 1, deverá existir um estado correspondente S^i_2 na máquina 2. Portanto, se na máquina 1 houver a transição de um estado i para outro estado j:

$$S^i_1 \longrightarrow S^j_1$$

então deverá existir a transição:

$$S^i_2 \longrightarrow S^j_2$$

na máquina 2.

Esta tarefa, pode não ser fácil de ser realizada devido às diferenças de arquitetura entre as duas. Quanto maior a diferença entre as arquiteturas, maior se torna o problema de estabelecer-se uma correspondência exata entre todas as características das duas máquinas. O projetista do emulador deve tentar aproximar da melhor forma possível o estado equivalente da hospedeira ao estado original da hóspede. Alguns exemplos dessas dificuldades são:

a) A tradução de uma operação aritmética numa máquina com as representações + 0 e - 0, em outra máquina que possue apenas a representação + 0 pode-se tornar difícil, caso o programador da primeira máquina queira atribuir significados diferentes a + 0 e - 0.

b) Podem ocorrer problemas quando a amplitude diferente de números nas máquinas 1 e 2. Por exemplo, se a máquina 1 tiver representações de números de 24 bits, e a máquina 2, de 16 bits, a emulação da soma de dois números de 24 bits pode tornar-se pouco eficiente na máquina 2, pois os números devem ser inicialmente truncados, trabalhando-se com as partes menos significativas de 16 bits, e logo após, com as partes mais significativas, de 8 bits, levando-se em conta o "vai-um" resultante da soma das partes menos significativas.

Nesse tipo de solução, o "vai-um" final deverá ser detetado no meio da palavra de 16 bits, o mesmo acontecendo com o transbordamento. Isto poderá ser feito por um microprograma, porém é uma operação muito lenta. Outra solução é a deteção por hardware, isto é, um circuito lógico especial adaptado para determinar o "vai-um" e transbordamento no meio da palavra de 16 bits.

c) A emulação de uma máquina com endereçamento de operações a bytes, quando efetuada numa máquina de palavra maior, acarreta o problema da separação de campos de 8 bits, o que pode tornar-se ineficiente se for usada a técnica de deslocamento de 8 bits.

Existem várias outras dificuldades que podem ser levantadas quando da emulação de uma máquina em outra, tais como, emulação de máquinas que usam representação numérica diferente da binária (decimais, BCD, hexadecimais, etc.), perda de eficiência em tempo de execução, etc. Essas dificuldades são crescentes com a diferenciação crescente entre as máquinas hóspedes e hospedeiras, podendo chegar a ser totalmente impraticável tal implementação.

Por outro lado, quanto mais compatíveis forem as máquinas, mais eficiente será a emulação, podendo até um programa ser rodado mais eficiente e rapidamente na hospedeira do que na hóspede. Além disso, quanto mais recursos, isto é, quanto mais características internas em hardware a hospedeira possuir, tanto mais ela servirá para abrigar um emulador.

No que se refere a operações de entrada e saída e interrupções, os problemas são um pouco mais complexos. Normalmente estas acarretam diferentes níveis de interrupção, o atendimento e tratamento de cada um nas duas máquinas, dispositivos e interfaces com programações diferentes. Se a máquina hospedeira tiver um sistema muito rígido de operação, isto é, se os canais forem muito automáticos e pouco flexíveis, a compatibilidade torna-se mais difícil. O programa original poderia também usar periféricos com características diferentes do que estão conectados à hospedeira, o que poderia requerer alguma alteração nas interfaces.

De uma maneira geral, os pontos a serem estudados tanto na máquina hóspede como na hospedeira quando da definição de um emulador microprogramado, são:

Memória principal	Comprimento da palavra (bits) Tamanho da memória (palavras)
Registradores	Reg. de propósito geral . número . comprimento (bits) Registradores especiais . número . função . comprimento (bits)
Dados	Tipos Formatos Convenções
Conjunto de instruções	Número de formatos Estrutura dos formatos . tipos de endereçamentos . modos de endereçamentos . cálculo de endereço efetivo Especificação detalhada de campos Códigos de operações Definições
Outras especificações especiais	

Figura VII.2 - Características das Máquinas Hóspede e Hospedeira

De uma maneira geral as informações a serem emuladas na hospedeira são as seguintes:

- informações de controle e estado da hóspede;
- dados da hóspede;
- instruções da hóspede.

Tanto na memória principal, como nos registradores, essas informações são indistingüíveis, isto é, são tratados pela hospedeira como configurações em bits e usadas convenientemente por solicitação do programa emulado.

a) Controle e Estado

As informações do controle e estado da hóspde estão associadas a um subconjunto de registradores da hóspede, tanto de propósito geral, como os especiais. É importante notar que pode-se eventualmente aproveitar registradores hospedeiros para as mesmas funções que os registradores hóspedes, caso ambos sejam equivalentes. Por exemplo, o registrador de endereços hospedeiro pode representar o registrador de endereços hóspede, desde que tenham o mesmo comprimento em bits. O mesmo pode ocorrer com o contador de instruções, acumuladores, bits de extensão ("vai-um"), transbordamento, delimitadores (bases e limites de segmentos de programa ou dados), etc.

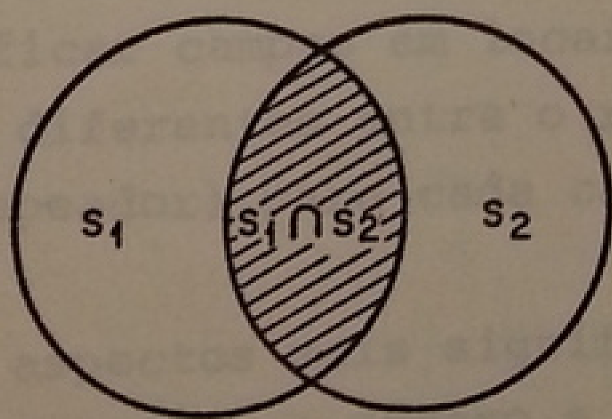


Figura VII.3 - Elementos equilaventes de S_1 e S_2

No diagrama acima, a intersecção do conjunto S_1 (características da máquina S_1) com o conjunto S_2 (características da máquina S_2), $S_1 \cap S_2$, significa elementos equivalentes. No caso da emulação de S_2 em S_1, o resto do conjunto de S_2 pode ser representado em outros recursos de S_1; portanto, observa-se que é interessante ter-se na hospedeira um número de recursos maior ou pelo menos igual ao dos recursos de mesmo tipo da hóspede.

b) Dados

Os possíveis tipos de dados podem incluir representações binárias fixas, de ponto flutuante, decimal e booleanas . Os formatos dos mesmos devem ser levados em conta, principalmente no que se refere a posições dos bits de sinal, característica e mantissa para ponto flutuante, representação de dados decimais, convenções aritméticas tais como complemento de 1, complemento de 2, sinal e amplitude, etc.

c) Conjunto de Instruções

A definição da máquina se completa com informações detalhadas acerca do conjunto de instruções. Tanto a sintaxe quanto a semântica da linguagem de máquina devem ser definidas. A sintaxe, nesse caso, se limita aos formatos das instruções, e para cada formato, a semântica se resume no significado de cada campo de instrução. A complexidade do emulador cresce com o número de formatos diferentes, pois para cada formato, o circuito interpretador precisa decodificar campos em locais diferentes, o que pode implicar em ligações diferentes entre o registrador de instruções e o interpretador (mapeador) para cada campo.

Um dos aspectos mais significativos do formato de uma instrução a ser emulada é o número e tipo de endereços contidos ou indicados pela instrução, já que a presença de um endereço

implica numa operação de busca ou armazenamento na memória principal. Na maior parte das instruções, os endereços são em número de zero a três. Tais endereços podem ser ponteiros a outras posições de memória (endereçamento indireto) que contém operandos, ou indicam a próxima instrução a ser executada.

VII.5 TÉCNICAS DE EMULAÇÃO

De uma maneira geral, o processo da emulação de uma instrução divide-se em três fases principais:

- Fase de Busca
- Fase de Cálculo de Endereço Efetivo
- Fase de Execução.

Pode-se observar que estas fases não diferem das fases de execução de uma instrução da própria máquina hospedeira, sendo esse processo bem geral, no sentido de poder ser aplicado a qualquer tipo de instrução de qualquer tipo de máquina hóspede. Eventualmente, pode ocorrer a omissão da Fase de Cálculo de Endereço Efetivo, como por exemplo no caso de uma instrução com operandos imediatos, explícitos na instrução.

Normalmente, essas três fases podem ser divididas em duas ou mais subfases. Estas subfases podem eventualmente ocorrer mais de uma vez como acontece com a busca de dois operandos, e podem não seguir necessariamente a ordem da figura VII.4.

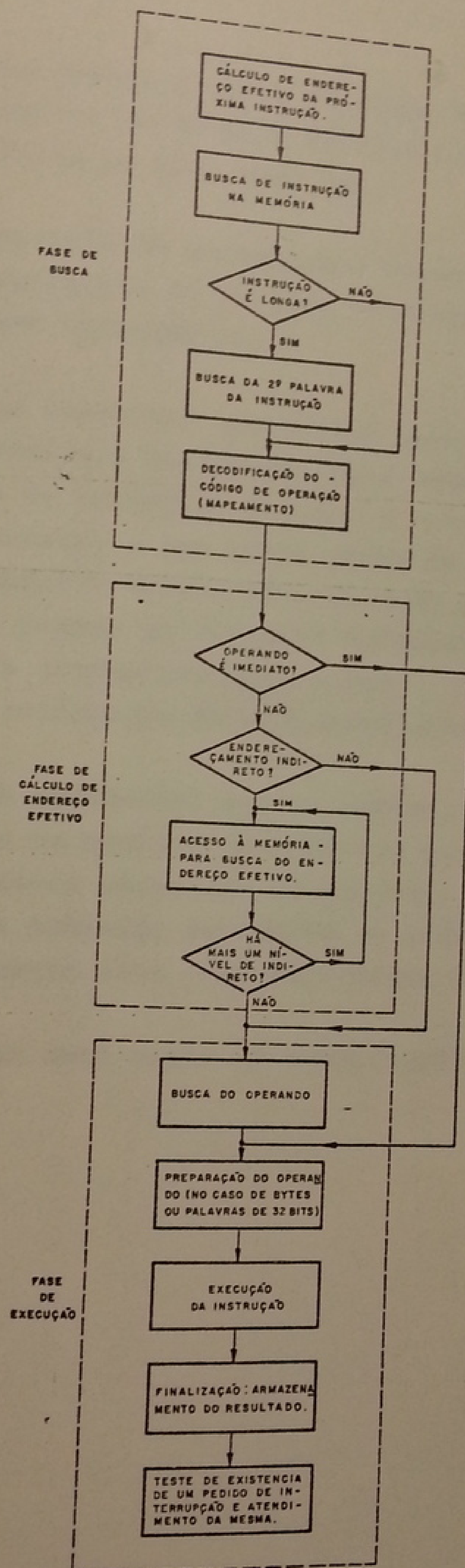


Figura VII.4 - Subfases de Emulação

Pode-se observar que o diagrama acima é bastante modular podendo-se omitir ou inserir módulos para emulação de características específicas de cada máquina.

A construção de um emulador requer basicamente duas considerações: o mapeamento da hóspede na hospedeira, e a elaboração dos microprogramas necessários.

Deve-se ressaltar que certas características de hardware podem ser adicionadas à hospedeira para aumentar a velocidade de execução, quando da emulação. O custo é evidentemente medido pelo acréscimo de circuitos e uma certa perda de generalidade. Por exemplo, o sistema IBM/370 Modelo 145, do modo como é implementado no processador 3145 possui um circuito decodificador do código de operação que executa automaticamente o desvio para a micro-rotina de execução, sem a utilização de microinstruções.

Existem emuladores que se servem, por outro lado, de tabelas armazenadas na memória de controle que facilitam tal decodificação sem circuitos especiais de hardware. O código de operação é comprado com as entradas da tabela, e cada código corresponde a um endereço de desvio para o microprograma de execução da instrução.

A seguir será visto um exemplo prático de emulação.

VIII - EXEMPLO DE EMULAÇÃO - SISTEMA IBM 1130

Nesse capítulo, será mostrada uma técnica de emular as instruções do sistema IBM 1130 Modelo 2D no minicomputador G-10.

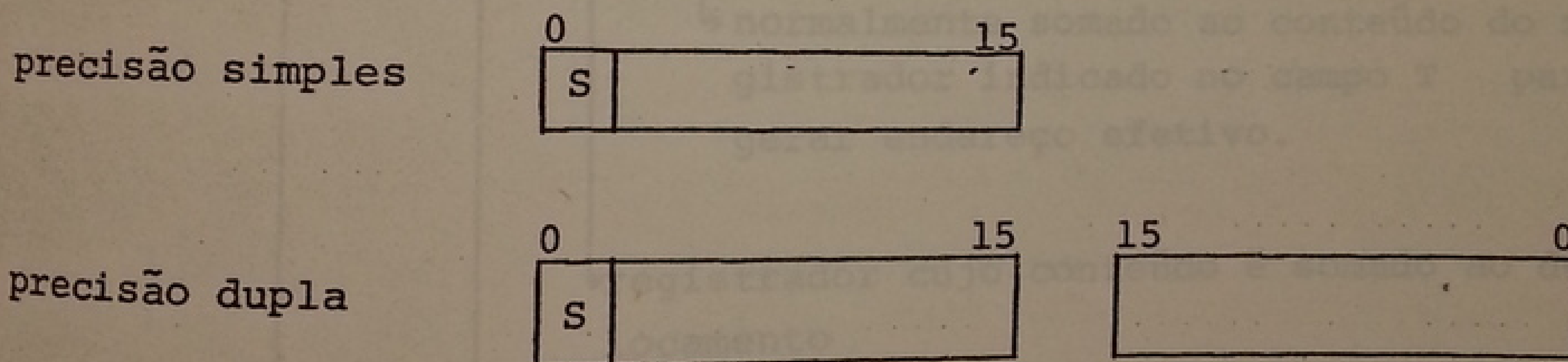
Primeiramente, é feito um resumo das principais características do IBM 1130, e são feitos comentários sobre elementos equivalentes em ambas as máquinas, e as possíveis representações de elementos diferentes na hospedeira.

VIII.1 CARACTERÍSTICAS PRINCIPAIS DO IBM 1130/2D [21]

a) Formato dos dados

Os dados podem ser de precisão simples (16 bits) ou dupla (32 bits), portanto totalmente compatíveis com o tamanho das palavras da hospedeira. Em ambas as máquinas os bits à esquerda são mais significativos.

O bit de sinal é o último à esquerda, tanto na precisão simples como na dupla.



A restrição nos dados duplos é que a palavra mais significativa deve estar localizada num endereço par da memória. Esse cuidado, porém, está a cargo do programador.

Os números negativos estão em complemento de 2.

b) Memória principal

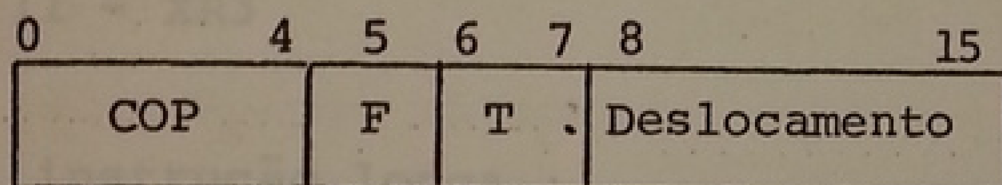
A capacidade máxima de memória do 1130 é 32K palavras, compatível com a hospedeira, que pode ter até 64K palavras.

As três primeiras posições da memória correspondem a três indexadores: XR1, XR2 e XR3.

c) Formato das instruções

O 1130 possui dois formatos de instruções: as curtas (16 bits) e as longas (32 bits):

Instruções curtas:



Deslocamento → normalmente somado ao conteúdo do registrador indicado no campo T para gerar endereço efetivo.

T → registrador cujo conteúdo é somado ao deslocamento

00 - Instruction Address Register (IAR)
01 - XR1
10 - XR2
11 - XR3

F → 0 - instrução curta
1 - instrução longa

COP → código de operação

Figura VIII.1 - Formato das Instruções Curtas do IBM 1130

Instrução longa:

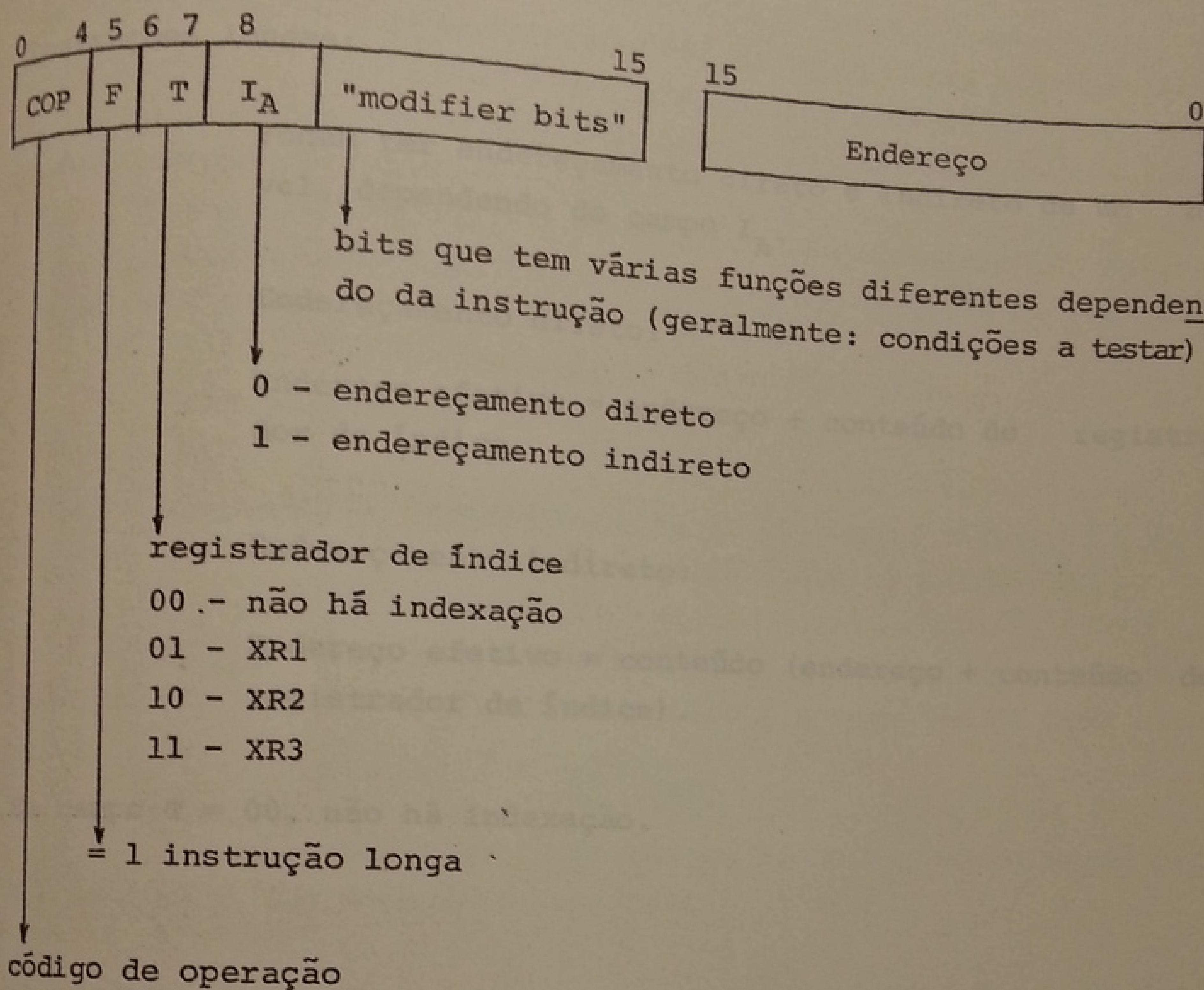


Figura VIII.2 - Formato das Instruções Longas do IBM 1130

d) <u>Endereçamento</u>

Instruções curtas:

Endereço efetivo = deslocamento + conteúdo do registrador especificado em T

Cabe aqui ressaltar que o registrador de endereço das instruções é atualizado durante a execução da instrução corrente,

para apontar a instrução logo a seguir.

Instruções longas:

Podem ter endereçamento direto e indireto de um nível, dependendo do campo I_A.

Endereçamento direto:

Endereço efetivo = endereço + conteúdo de registrador de índice.

Endereçamento indireto:

Endereço efetivo = conteúdo (endereço + conteúdo do registrador de índice).

Se campo T = 00, não há indexação.

VIII.2 EMULAÇÃO DAS INSTRUÇÕES DO IBM 1130

Analisando-se os formatos das instruções e o esquema de endereçamento, propõe-se aqui uma técnica de mapeamento com o objetivo de que a tradução dos códigos de operação e outras informações contidas na instrução sejam executadas o mais rapidamente possível.

Para se obter uma determinada eficiência na emulação, pode-se fazer algumas modificações nos circuitos de tal maneira a facilitar a microprogramação das instruções.

Entretanto, a identidade do G-10 deve ser conservada, e ao mesmo tempo possibilitada uma emulação eficiente.

Justifica-se, portanto, quando do projeto de um computador microprogramado, a previsão da possibilidade dessa máquina poder ser facilmente adaptada para operações de emulação. Esta previsão se reflete na organização interna dos componentes lógicos da hospedeira de tal maneira que as eventuais mudanças do hardware por ocasião da implantação de um emulador se concentrem num só local, por exemplo numa só placa. Assim, seria possível ter-se dois sistemas operando numa só máquina, alternando-se entre o uso de um ou outro mediante um simples chaveamento.

Este chaveamento poderia ser de dois tipos:

a) Manual. A determinação do sistema a vigorar é feita mediante a troca da placa com as modificações em hardware.

b) Programado. O chaveamento é feito por instruções especiais da hospedeira, que possibilitem a operação em modo normal ou emulação.

A seguir serão avaliadas as modificações do hardware necessárias para a implantação do emulador.

Vários campos da instrução da hóspede são diferentes dos da hospedeira, o que implica numa interpretação diferente dos mesmos por parte do mapeador. Felizmente, nesse caso, os códigos de operação tanto do G-10 como do IBM 1130 ocupam os mesmos campos na instrução, portanto, não seriam necessárias mudanças em hardware no mapeador, a não ser quanto aos endereços de mapeamento contidos nas memórias "ler-somente".

Existe um outro método de identificação de informações contidas na instrução, que emprega exclusivamente a microprogramação para a extração e análise individual para cada campo da instrução. Este mapeamento microprogramado, porém, é muito lento, podendo acarretar um tempo de execução da instrução bem maior do que o tempo na máquina original.

a) Análise dos Campos das Instruções

COP: O campo de código de operações de 5 bits (I (0-4)) pode ser a entrada de um dos mapeadores, cuja saída pode ser selecionada pela micro-ordem FOR 1.

F: O campo que especifica se a instrução é curta ou longa pode ser diretamente testado por micro-ordens condicionais da hospedeira (esta possui um teste análogo).

I: Este campo de 2 bits (I (6-7)) pode endereçar os registradores R0, R1, R2 e R3 diretamente com pequenas alterações no hardware (eles podem ser endereçados pelo conjunto de micro-ordens SRED do campo A e ORGM do campo MUXE). R0 representaria o IAR nesse caso.

IA: Este campo que indica endereçamento indireto, pode ser testado diretamente por micro-ordens condicionais. (Existe teste análogo nas instruções da hospedeira).

"Modifier bits" : Este campo constitui uma máscara que especifica condições de estado a serem testadas.

"Modifier bits" (=1)	Especificação
15	Transbordamento = 0
14	Extensão = 0
13	Operando no acumulador é par
12	Opernado no acumulador é positivo
11	Operando no acumulador é negativo
10	Operando no acumulador é zero

Figura VIII.3 - "Modifier bits" do IBM 1130

A instrução condicional especificando uma ou mais condições a serem testadas não é executada no caso em que pelo menos uma das condições seja verdadeira. Caso contrário, a instrução é executada normalmente. Esse campo é basicamente usado nas interrupções de desvio.

A hospedeira possui no registrador de estado do processador, todos os bits acima mencionados, exceto bits distintos para acumulador positivo e negativo. Esse problema é facilmente resolvido pois o registrador de estado da hospedeira possui 8 bits disponíveis para testes, e no caso, um dos bits é o complemento do outro, e portanto há folga suficiente para implementar-se nesse registrador mais um bit para representar o acumulador negativo, por exemplo.

As condições do registrador de estado também podem ser rearranjadas sem muita dificuldade para que sua ordem coincida com o campo Modifier bits do 1130, de tal maneira que a comparação do registrador de estado e desse campo seja fácil.

Outra maneira de comparar os bits da máscara com os so registrador de estado seria uma comparação seqüencial de bit a bit, o que, porém, resultaria na execução mais lenta da instrução.

b) <u>Mapeador</u>

O mapeador para as instruções do IBM 1130 possui duas memórias "ler-somente", M2 e M3, analogamente ao mapeador da hospedeira. M2 é selecionado com a micro-ordem FOR 2, e tem a função de mapear uma micro-rotina de cálculo de endereço efetivo. M3 é selecionado com as micro-ordens EEX e FOR 2, e tem a função de mapear uma micro-rotina de execução.

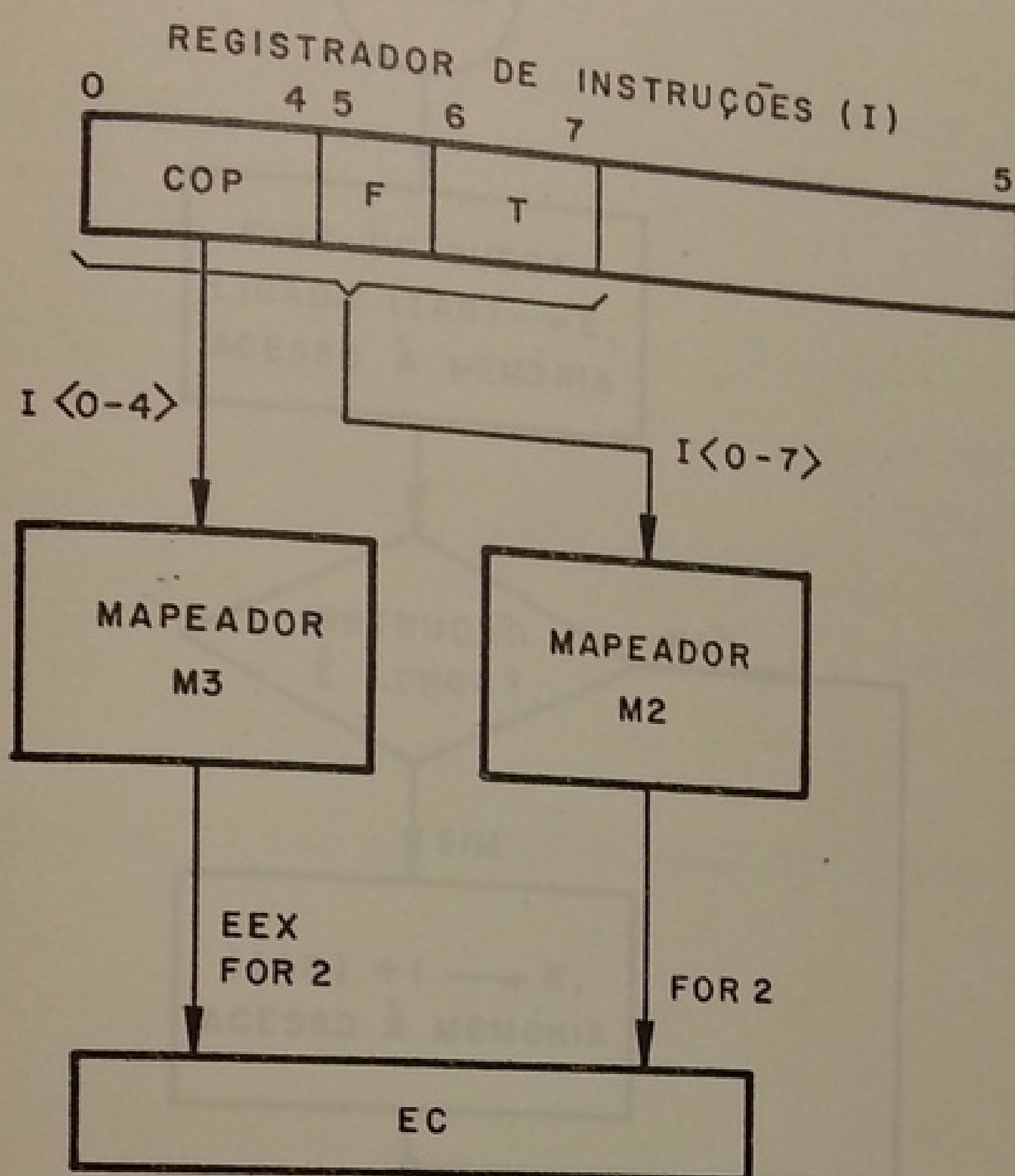


Figura VIII.4 - Mapeador do Emulador

c) Fases da Microprogramação

c.1) Busca da instrução

A micro-rotina geral de busca, residente na posição zero da memória de controle, teria o seguinte diagrama de blocos:

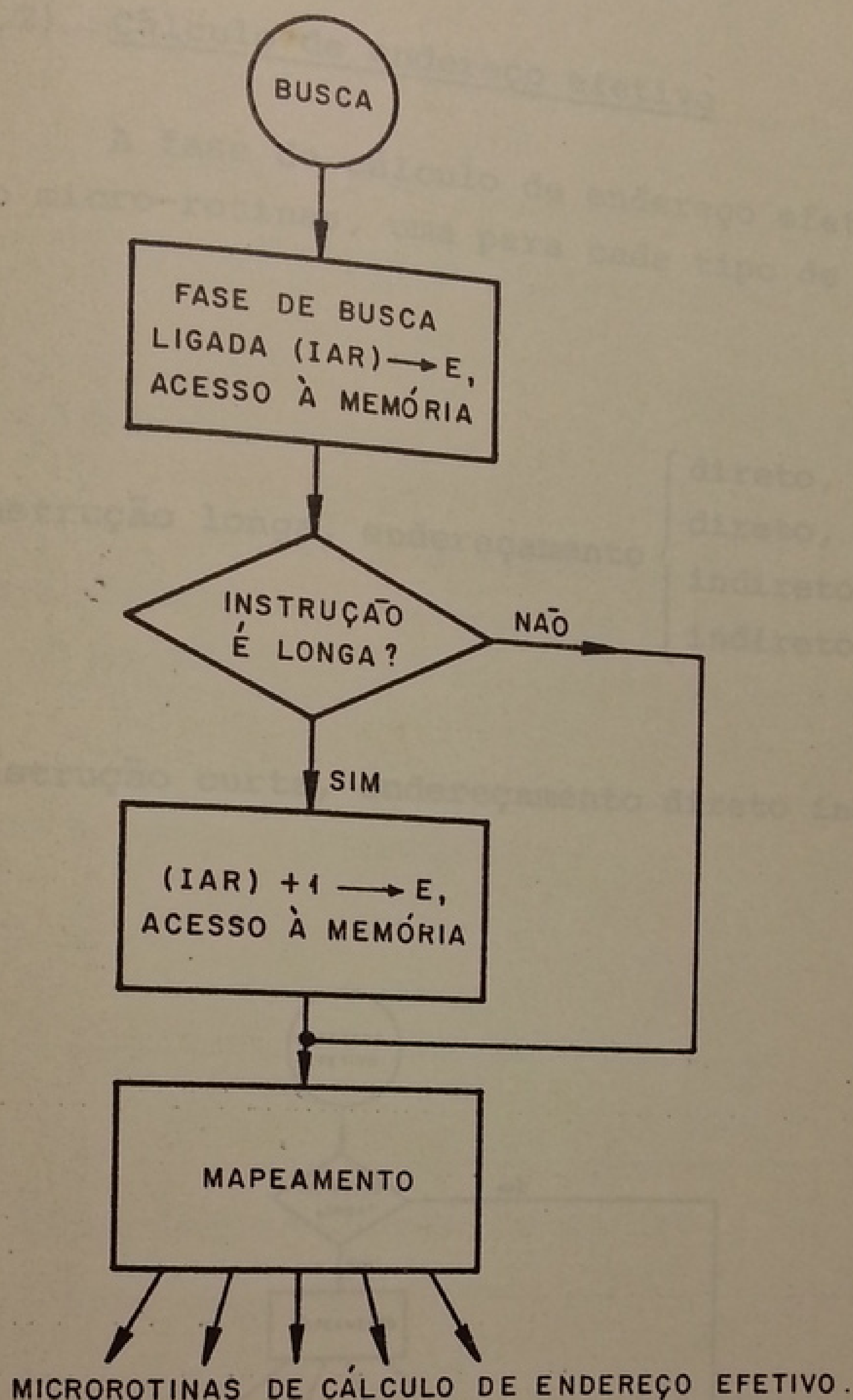


Figura VIII.5 - Busca de Instrução IBM 1130

As condições finais dessa micro-rotina são:

- 1a. palavra da instrução no registrador I
- 2a. palavra, no registrador D
- IAR (em R0) aponta a última posição acessada na memória.

c.2) Cálculo de endereço efetivo

A fase de cálculo de endereço efetivo é representada por cinco micro-rotinas, uma para cada tipo de endereçamento:

- instrução longa, endereçamento
  - direto, não indexado
  - direto, indexado
  - indireto, não indexado
  - indireto, indexado

- instrução curta, endereçamento direto indexado.

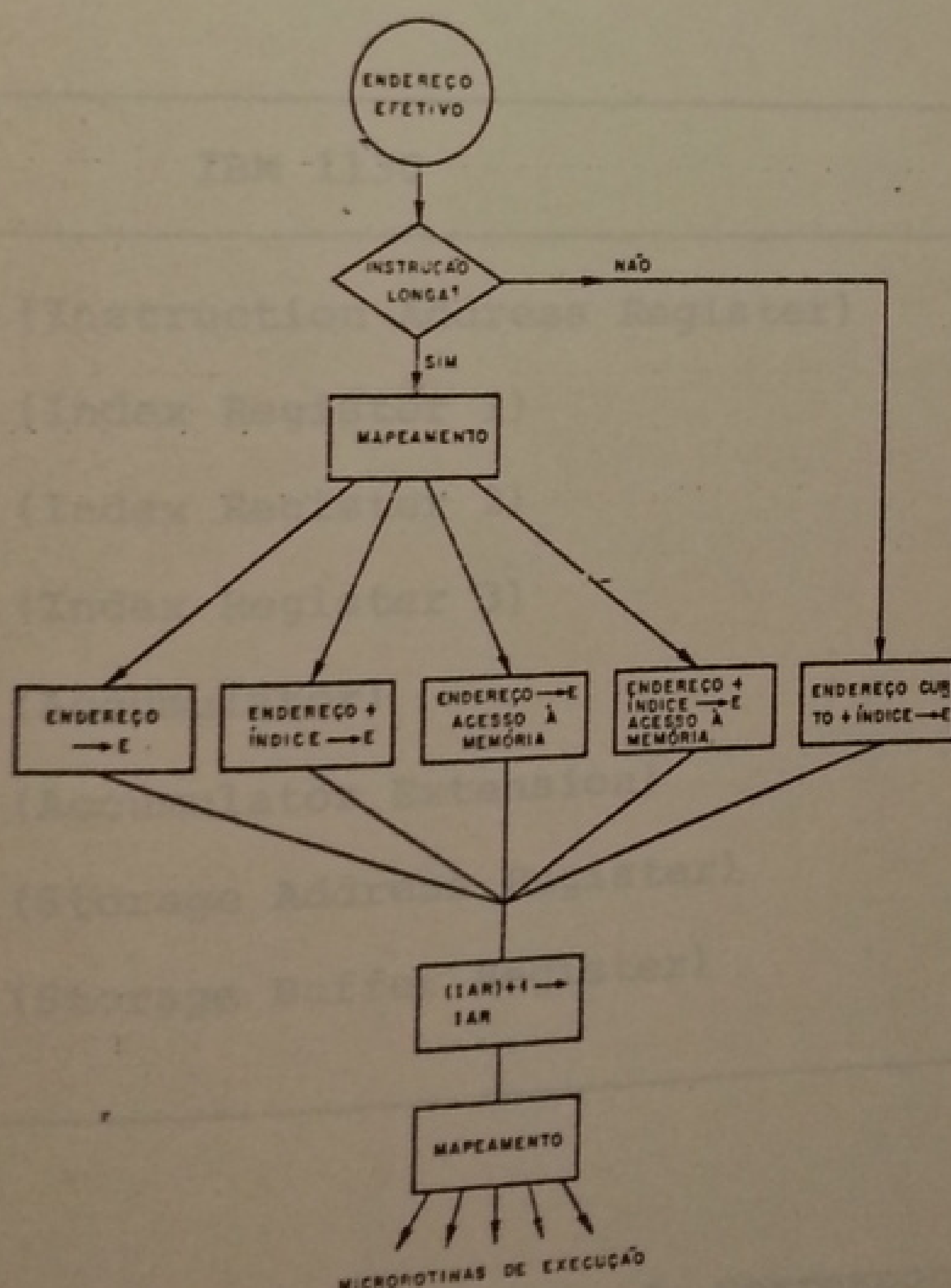


Figura VIII.6 - Cálculo de Endereço Efetivo

As condições finais em todas as micro-rotinas são:

- 1a. palavra da instrução no registrador I
- endereço efetivo no registrador E

Execução

As micro-rotinas de execução das instruções de carga e armazenagem, aritméticas, de deslocamento e de desvio encontram-se nas listagens do apêndice C, exceção feita às de entrada e saída.

Foram estabelecidas as seguintes correspondências entre registradores da hóspede e hospedeira:

IBM 1130	Hospedeira
IAR (Instruction Address Register)	R0
XR1 (Index Register 1)	R1
XR2 (Index Register 2)	R2
XR3 (Index Register 3)	R3
ACC (Accumulator)	R6
EXT (Accumulator Extension)	R7
SAR (Storage Address Register)	E
SBR (Storage Buffer Register)	D

Figura VIII.7 - Tabela de Correspondência entre Registradores da Hóspede e Hospedeira

VIII.3 TEMPOS DE EMULAÇÃO

Com o objetivo de ter uma noção sobre a eficiência da emulação das instruções do IBM 1130 no G-10, levando em conta o critério de avaliação segundo tempo de processamento na UCP, foi feita uma comparação entre tempos de execução na máquina hóspede original e a hospedeira.

A tabela da figura VIII.8 foi extraída da referência [21] e consta de uma lista de tempos médios das instruções, em microsegundos (us).

Instrução	Mnem.	Instr. curta (F=0) T = 00	Instr. curta (F=0) T=01,10,11	Instr. longa (F=1) T = 00	Instr. longa (F=1) T=01,10,11
LOAD ACC	LD	7,6	11,2	10,8	14,8
LOAD DOUBLE	LDD	11,2	14,9	14,4	18,0
STORE ACC	STO	7,6	11,2	10,8	14,8
STORE DOUBLE	STD	11,2	14,9	14,4	18,0
LOAD INDEX	LDX	4,5	7,2	7,2	11,8
STORE INDEX	STX	7,6	11,2	11,8	15,4
LOAD STATUS	LDS	3,6	3,6	-	-
STORE STATUS	STS	7,6	11,2	10,8	14,8
ADD	A	8,0	11,7	11,2	15,3
ADD DOUBLE	AD	12,2	15,8	15,3	19,3
SUBTRACT	S	8,0	11,7	11,2	15,3
SUBTRACT DOUBLE	SD	12,2	15,8	15,3	19,3
MULTIPLY	M	25,7	29,3	29,3	32,9
DIVIDE	D	76,6	79,6	79,6	83,2
AND	AND	7,6	11,2	10,8	14,8
OR	OR	7,6	11,2	10,8	14,8
EXCLUSIVE OR	EOR	7,6	11,2	10,8	14,8
NO OPERATION	NOP	3,6	-	-	-
SHIFT LEFT ACC, 00	SLA	3,2	6,4	-	-
SHIFT LEFT ACC AND EXT,10	SLT	3,2	6,4	-	-
SHIFT LEFT AND COUNT ACC,01	SLCA	3,2	6,4	-	-
SHIFT LEFT AND COUNT ACC AND EXT, 11	SLC	3,2	6,4	-	-
SHIFT RIGHT ACC,00 OR 01	SRA	3,2	6,4	-	-
SHIFT RIGHT ACC AND EXT,10	SRT	3,2	6,4	-	-
ROTATE RIGHT, 11	RTE	3,2	6,4	-	-
BRANCH AND STORE IAR	BSI	7,6	11,2	10,8	14,8
BRANCH OR SKIP ON CONDITION	BSC	3,6	3,6	7,2	11,2
MODIFY INDEX AND SKIP	MDX	4,5	11,2	18,5	18,5
WAIT	WAIT	3,6	3,6	-	-

IBM 1130

Figura VIII.8 - Tempos Médios das Instruções do IBM 1130

Esses tempos não levam em conta endereçamento indireto, caso em que deveriam ser acrescentados 3,6 us ao tempo da instrução na tabela.

A partir das listagens no apêndice C, pode-se construir a tabela da figura VIII.9 correspondente aos tempos médios de execução das instruções na hospedeira (G-10). Esses tempos foram computados pela contagem de números de ciclos de UCP, tempo de acesso à via comum e ciclos de memória para cada instrução. O ciclo da UCP é de 300 ns e o da memória de núcleos de ferrite é de 850 ns. O tempo de acesso à via comum do G-10 foi considerado igual ao ciclo de UCP.

A tabela da figura VIII.9 contém as velocidades de execução das instruções no G-10, relativas às do IBM 1130. Assim, por exemplo, se a instrução LD é executada em 7,6 us no IBM 1130, e o é em 2,6 us no G-10, sua velocidade relativa é aproximadamente 3.

G-10

Mnem.	Instrução curta (F = 0)		Instrução longa (F = 1)	
	T = 00	T = 01,10,11	T = 00	T=01,10,11
LD	2,6	2,6	3,0	3,3
LDD	3,9	3,9	4,3	4,6
STO	2,9	2,9	3,3	3,6
STD	3,9	3,9	4,3	4,6
LDX	3,2	3,2	3,9	4,2
STX	2,9	2,9	3,3	3,6
LDS	2,2	2,2	-	-
STS	4,7	4.7	5,1	5,4
A	2,6	2,6	3,0	3,3
AD	4,5	4,5	4,9	5,2
S	2,6	2,6	3,0	3,3
SD	4,5	4,5	4,9	5,2
M	9,0	9,0	9,3	9,6
D	25,0	25,0	25,4	25,7
AND	2,6	2,6	3,0	3,3
OR	2,6	2,6	3,0	3,3
EOR	2,6	2,6	3,0	3,3
NOP	3,0	-	-	-
SLA	4,3	4,3	-	-
SLT	4,3	4,3	-	-
SLCA	4,3	6,4	-	-
SLC	4,3	6,4	-	-
SRA	4,6	4,6	-	-
SRT	5,2	5,2	-	-
RTE	5,2	5,2	-	-
BSI	4,4	4,4	5,1	5,4
BSC	3,1	3,1	3,5	3,8
MDX	2,8	4,3	4,6	5,0
WAIT	2,8	2,8	-	-

Figura VIII.9 - Tempos Médios de Emulação no G-10

Mnem.	Velocidades Relativas			
	Instrução curta (F = 0)		Instrução longa (F = 1)	
	T = 0	T = 01,10,11	T = 00	T=01,10,11
LD	3,0	4,3	3,6	4,5
LDD	2,9	3,8	3,3	3,9
STO	2,6	3,8	3,3	3,9
STD	2,9	3,8	3,3	3,9
LDX	1,4	2,3	1,8	2,8
STX	2,6	3,9	3,6	4,3
LDS	1,6	,16	-	-
STS	1,6	2,4	2,1	2,7
A	3,1	4,5	3,7	4,6
AD	2,7	3,5	3,1	3,7
S	3,1	3,5	3,7	4,6
SD	2,7	3,5	3,1	3,7
M	2,9	3,3	3,2	3,4
D	3,0	3,2	3,1	3,2
AND	2,9	4,3	3,6	4,5
OR	2,9	4,3	3,6	4,5
EOR	2,9	4,3	3,6	4,5
NOP	1,2	-	-	-
SLA	0,7	1,5	-	-
SLT	0,7	1,5	-	-
SLCA	0,7	1,0	-	-
SLC	0,7	1,0	-	-
SRA	0,7	1,4	-	-
SRT	0,6	1,2	-	-
STE	0,6	1,2	-	-
BSI	1,7	2,5	2,1	2,7
BSC	1,7	1,7	2,1	2,9
MDX	1,6	2,6	4,0	3,7
WAIT	1,3	1,3	-	-

Figura VIII.10 - Velocidades Relativas de Emulação

Observando as tabelas, nota-se que quase todas as instruções (exceto as de "shifts") tem tempo de execução menor na hospedeira do que na máquina original.

Nota-se também que, em geral, com o aumento da complexidade da instrução (curta ou longa, T = 00 ou T = 01, 10, 11) a velocidade de execução na hospedeira aumenta, relativamente à velocidade na máquina original. Com um nível de endereçamento indireto (para esse caso os tempos de execução não constam na tabela), a execução da instrução na hospedeira tem uma velocidade relativa ainda maior, pois o acréscimo no tempo corresponde a um ciclo de acesso à memória, de 3,6 us no IBM 1130, enquanto que na hospedeira é de aproximadamente 1,15 ms (acesso à via + ciclo de memória).

Uma conclusão a que se pode chegar é que, quanto a tempo de processamento de instruções na UCP, a emulação de um sistema construído com tecnologia mais antiga é vantajosa, desde que as características fundamentais tanto da hóspede, como da hospedeira, não sejam muito discrepantes.

No que se refere a tecnologia, pode-se dizer que um dos principais fatores é o ciclo de acesso à memória principal. Quanto maior este ciclo na máquina a ser emulada, mais eficiente se torna sua emulação numa máquina implementada com tecnologia mais recente.

Em particular, a máquina emulante ainda possui a característica adicional de poder executar operações no processador central paralelamente ao acesso à memória (desde que o registrador de dados não seja utilizado antes que este contenha o dado correto). Isto possibilita que algumas operações sejam transparentes ao processamento geral da instrução emulada, isto é, seu tempo de execução não é contado, mas sim, é englobado dentro do tempo de acesso à memória.

Estas características são todas importantes na computação geral de tempos de execução, principalmente para programas do tipo "CPU - bound" (tempo de processamento grande da UCP).

VIII.4 CRITÉRIOS DE AVALIAÇÃO DO EMULADOR

Para a avaliação da eficiência de uma máquina, o critério mais freqüentemente usado tem sido o de considerar o tempo de ciclo de memória e o tempo de uma adição. Esses dois parâmetros podem fornecer alguma informação útil, mas não são considerados bons para uma avaliação correta, porque eles ignoram o efeito causado pelas circunstâncias em que estão sendo processados os programas. Por exemplo, o ciclo de memória de uma máquina pode ser lento, porém o tempo de execução da instrução pode ser muito rápido, inclusive mais rápido do que uma instrução numa máquina com ciclo de memória muito pequeno. Assim também uma instrução de adição pode ser mais lenta numa máquina do que em outra, mas ser muito mais poderosa.

Existem outros métodos de avaliação, um dos quais leva em conta o "mix" de instruções. Um "mix" de instruções é uma lista ponderada das instruções de uma máquina. Essa ponderação (ou peso) associada a cada instrução é baseada na sua freqüência de execução, estimada pela observação de um programa em processamento. Portanto, para esse programa, pode-se dizer que seu "mix" é um indicador bastante realista para a avaliação.

O tempo de execução de uma instrução média pode ser calculado a partir da seguinte expressão:

$$T = \frac{1}{n} \sum_{i=1}^{n} f_i \, t_i \qquad (1)$$

onde: n é o número de instruções
f_i é a freqüência relativa da i-ésima instrução
t_i é o tempo de execução da instrução i.

Existe uma série de "mixes" padrão desenvolvidos para fins específicos. O mais conhecido é o "gibson mix", da figura VIII.11 [16].

Tipos de Instruções	Frequência por 100 instruções
Carga e Amarzenagem	31,2
Adição e Subtração	6,1
Comparações	3,8
Desvios	16,6
Adição e Subtração Flutuantes	6,9
Multiplicação Flutuantes	3,8
Divisão Flutuante	1,5
Multiplicação	0,6
Divisão	0,2
Deslocamentos	4,4
Operações Lógicas(E, OU, etc.)	1,6
Instruções que não usam registradores	5,3
Indexação	18,0

Figura VIII.11 - "Gibson Mix"

Outros métodos de avaliação mais atuais incluem processamento de programas tipo "benchmark" padrão, modelos matemáticos e simulações. De uma maneira geral, porém em todos os casos há variações nas medidas, dependendo da arquitetura da máquina, sistema operacional e aplicações.

No caso de um emulador, sua eficiência pode ser estimada pelo fator P de potência:

$$P = \frac{\tau_1}{\tau_2} \qquad (2)$$

onde: τ_1 é o tempo de processamento na máquina original.

τ_2 é o tempo de processamento de um programa na hospedeira.

Se $P < 1$, o emulador é menos eficiente que a hóspede.

Se $P \geqslant 1$, o emulador é tão bom ou melhor que a hóspede.

No caso do IBM 1130 emulado no G-10, observando as tabelas das figuras VIII.8, VIII.9 e VIII.11, e aplicando as fórmulas (1) e (2), pode-se chegar ao valor aproximado da potência do emulador:

$$P = 3,59$$

VIII.5 EMULAÇÃO DE INSTRUÇÕES DE ENTRADA E SAÍDA

A emulação do sistema de entrada e saída requer mais esforço do que no caso de instruções comuns da UCP. Isto porque o IBM 1130 possui uma filosofia diferente do esquema de entrada e saída do G-10, tanto no que se refere a atendimento de interrupções, como também dos próprios dispositivos.

Uma das maneiras de resolver esse problema é projetar e implementar interfaces para adaptar os dispositivos IBM no G-10 por hardware, atingindo assim uma compatibilidade quase total entre os dois sistemas. Estas modificações, porém, acarretam um esforço considerável em hardware, e restringem o sistema a usar apenas os dispositvos periféricos da IBM.

Para contornar esse fato, os emuladores existentes normalmente utilizam uma simulação em software das instruções de entrada e saída e interrupções em geral.

Este software constituiria um pequeno sistema monitor, encarregado de interpretar as instruções de entrada e saída da hóspede, ativando os periféricos da mesma, não sendo estes necessariamente da IBM. O sistema monitor também interpretaria os estados dos periféricos (ocupado ou livre, esperando reconhecimento e atendimento de interrupção, condições de erro, etc.), assim como interrupções da hóspede.

A emulação do sistema de entrada e saída não será abordada aqui, pois devido à sua complexidade, deve constituir um trabalho à parte.

IX - CONCLUSÕES

Neste trabalho foram apresentados alguns aspectos de microprogramação e problemas decorrentes de sua implementação no minicomputador G-10. A solução desses problemas não pode ser considerada ótima em todos os pontos, porém foi bastante adequada no contexto de implementação da arquitetura de uma máquina de propósito geral como o G-10.

Várias características na microprogramação como, por exemplo, o formato da microinstrução, mapeadores e otimização, foram definidos segundo os resultados práticos obtidos em tentativas sucessivas no projeto. Isto é, as falhas nas primeiras estruturas projetadas, tanto de hardware como de microprogramação serviram como base para o aprimoramento do projeto, de uma maneira iterativa.

O resultado a que se chegou é a percepção do fato de que microprogramar talvez possa ser mais difícil do que programar, no sentido de que a microinstrução é um ente programável mais complexo do que uma instrução de máquina, e requer um conhecimento mais aprofundado da estrutura interna do processador. É possível que por essa razão a prática da microprogramação por um usuário comum, ainda não esteja muito difundida, apesar de alguns sistemas como o HP 2100, possuírem documentação para tal.

A microprogramação, porém é muito útil sob o ponto de vista do projetista do hardware e do sistema, pois ela permite uma centralização e sistematização do projeto de uma máquina, tornando sua implementação e manutenção mais fáceis.

Algumas sugestões para pesquisas nesse campo seriam desenvolver métodos para geração de palavras de controle ótimas em termos de número de bits, ou tempo de execução, para uma sistema em fase de projetó, e também geração de testes automáticos de microprogramas sem falhas.

Para o engenheiro de sistemas, a microprogramação constitui um campo aberto para a implementação de características de

software em microcódigo, tornando certas tarefas mais automáticas e eficientes.

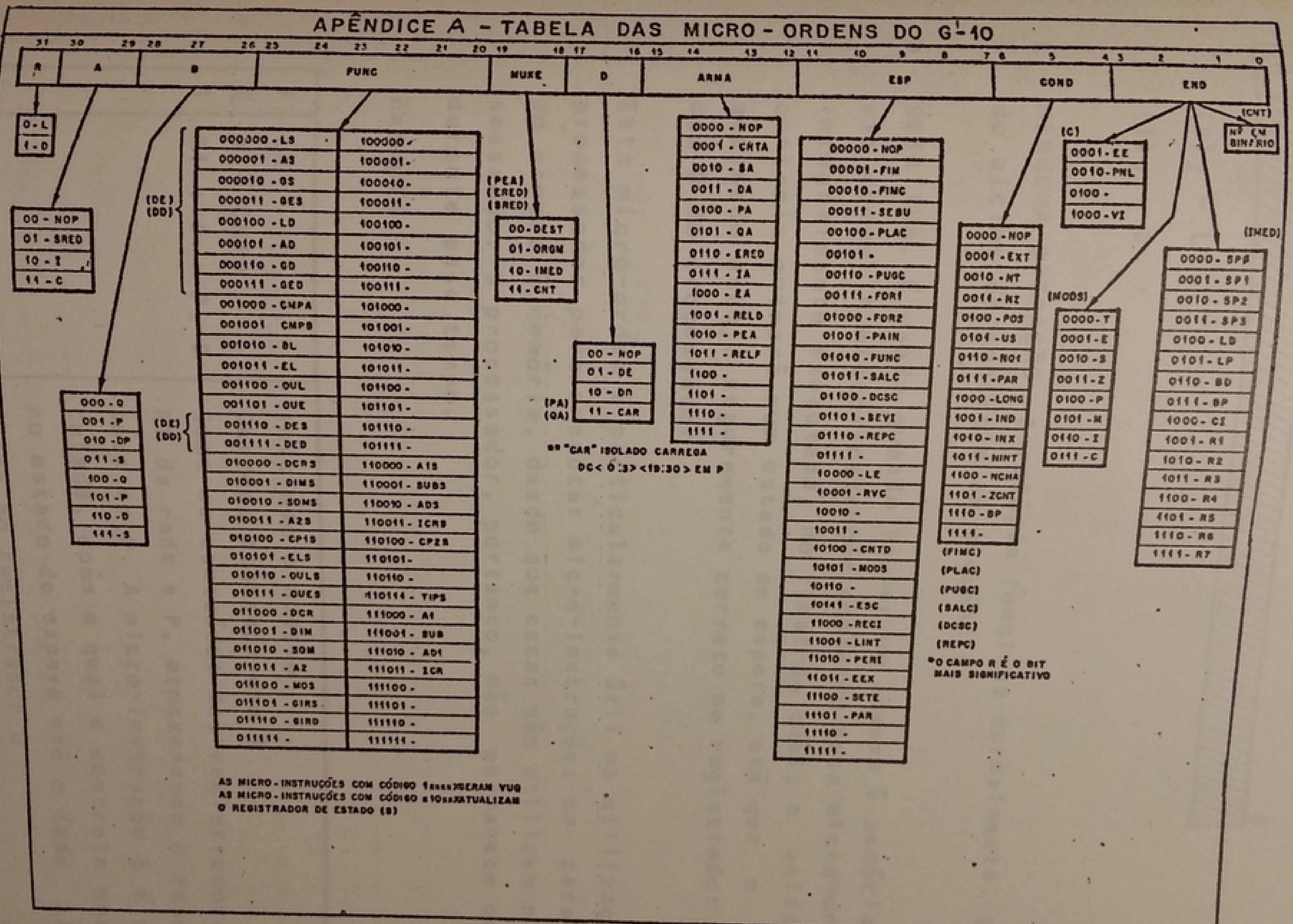
Exemplos para esta atividade são muitos: desenvolvimento de unidades de controle microprogramadas para discos, montadores automáticos [2], compiladores, sistemas de tempo compartilhado, microdiagnose, emulação, linguagens de alto nível (FORTRAN, ALGOL), etc. [5], como o que ocorre no sistema B1700 que facilita a implementação de linguagens de alto nível e compiladores, o IBM 370/145 que possui a opção de microdiagnose, e o desenvolvimento do emulador do sistema Burroughs 220 para o IBM 360/25.

Seria interessante, também, o desenvolvimento da microprogramação em microprocessadores, e terminais inteligentes. Atualmente há muito poucas referências sobre esse assunto pelo fato de que os microprocessadores constituem uma tecnologia muito recente.

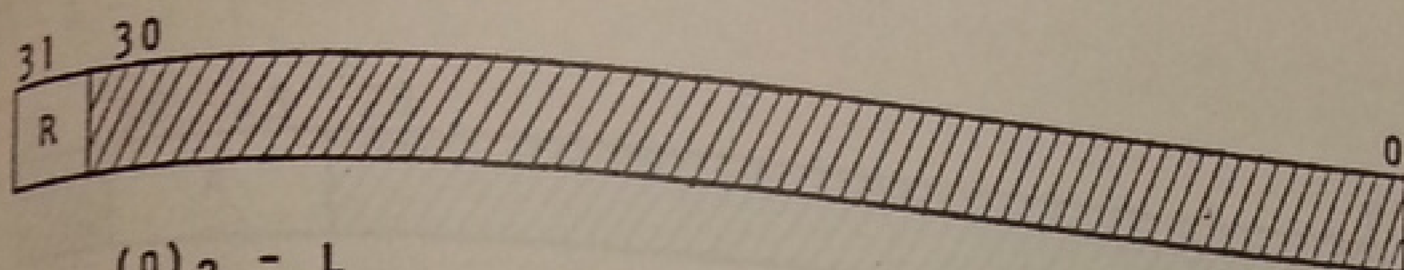
A elaboração de um sistema de vários emuladores funcionando em multiprogramação [13] também tem suscitado interesse, e é sugerida como tema para um futuro campo de pesquisa.

Finalmente, pode-se dizer que a maior utilidade da microprogramação no futuro, é fazer com que o custo cada vez mais baixo do hardware, associado às facilidades oferecidas pela microprogramação, levem a construções de sistemas de custo global menor, capazes de processar programas com grande eficiência, dentro das aplicações a que são destinados.

APÊNDICE A - CONJUNTO DE MICRO-ORDENS DO G-10


APÊNDICE A - TABELA DAS MICRO-ORDENS DO G-10
R
A
B
FUNC
MUXE
D
ARNA
ESP
COND
END
0-L
1-D
00-NOP
01-SRED
10-I
11-C
000-Q
001-P
010-DP
011-S
100-Q
101-P
110-D
111-S
(DE)
(DD)
000000-LS
000001-AS
000010-OS
000011-OES
000100-LD
000101-AD
000110-OD
000111-OED
001000-CMPA
001001 CMPB
001010-BL
001011-EL
001100-OUL
001101-OUE
001110-DES
001111-DED
010000-DCRS
010001-DIMS
010010-SOMS
010011-A2S
010100-CP1S
010101-ELS
010110-OULS
010111-OUES
011000-DCR
011001-DIM
011010-SOM
011011-A2
011100-MOS
011101-GIRS
011110-GIRD
011111-
100000-
100001-
100010-
100011-
100100-
100101-
100110-
100111-
101000-
101001-
101010-
101011-
101100-
101101-
101110-
101111-
110000-A1S
110001-SUBS
110010-ADS
110011-ICRS
110100-CP2S
110101-
110110-
110111-TIPS
111000-A1
111001-SUB
111010-AD1
111011-ICR
111100-
111101-
111110-
111111-
AS MICRO-INSTRUÇÕES COM CÓDIGO 1xxxxx GERAM VUG
AS MICRO-INSTRUÇÕES COM CÓDIGO x10xxx ATUALIZAM O REGISTRADOR DE ESTADO (S)
(PEA)
(ERED)
(SRED)
00-DEST
01-ORGM
10-IMED
11-CNT
00-NOP
01-DE
10-DD
11-CAR
(PA)
(QA)
** "CAR" ISOLADO CARREGA DC<0:3><19:30> EM P
0000-NOP
0001-CNTA
0010-SA
0011-DA
0100-PA
0101-QA
0110-ERED
0111-IA
1000-EA
1001-RELD
1010-PEA
1011-RELF
1100-
1101-
1110-
1111-
00000-NOP
00001-FIM
00010-FINC
00011-SEBU
00100-PLAC
00101-
00110-PUGC
00111-FOR1
01000-FOR2
01001-PAIN
01010-FUNC
01011-SALC
01100-DCSC
01101-SEVI
01110-REPC
01111-
10000-LE
10001-RVC
10010-
10011-
10100-CNTD
10101-MODS
10110-
10111-ESC
11000-RECI
11001-LINT
11010-PERI
11011-EEX
11100-SETE
11101-PAR
11110-
11111-
0000-NOP
0001-EXT
0010-NT
0011-NZ
0100-POS
0101-US
0110-NO1
0111-PAR
1000-LONG
1001-IND
1010-INX
1011-NINT
1100-NCHA
1101-ZCNT
1110-BP
1111-
(FINC)
(PLAC)
(PUGC)
(SALC)
(DCSC)
(REPC)
*O CAMPO R É O BIT MAIS SIGNIFICATIVO
(C)
0001-EE
0010-PNL
0100-
1000-VI
(MODS)
0000-T
0001-E
0010-S
0011-Z
0100-P
0101-M
0110-I
0111-C
(CNT)
Nº EM BINÁRIO
(IMED)
0000-SPØ
0001-SP1
0010-SP2
0011-SP3
0100-LD
0101-LP
0110-BD
0111-BP
1000-CI
1001-R1
1010-R2
1011-R3
1100-R4
1101-R5
1110-R6
1111-R7

A.1 CAMPO R



$(0)_2$ - L

$(1)_2$ - D

L NENHUMA OPERAÇÃO. O relógio funciona normalmente, gerando ciclos de controle em sequência.

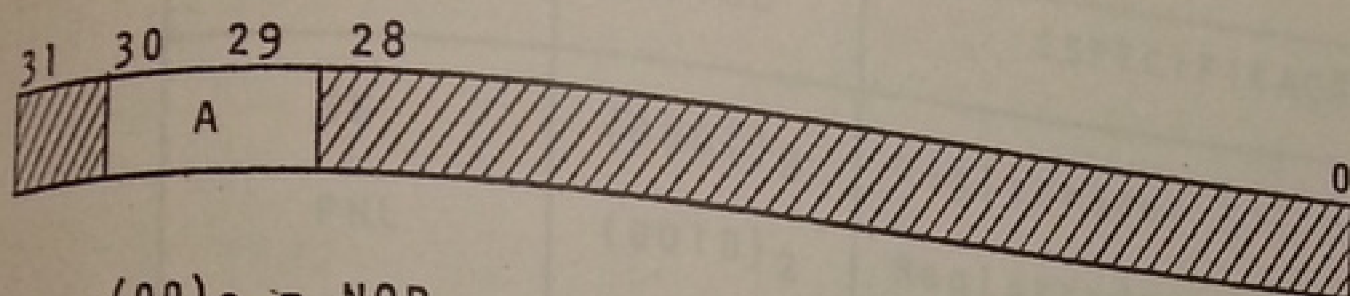
D DESATIVA. É usada quando houver um acesso à memória ou à via comum. A micro-instrução contendo essa micro-ordem é integralmente executada, no final da qual, a unidade de controle entrará num estado de espera, até que o dado acessado esteja seguramente correto no registrador de dados D do processador.

Esta micro-ordem é particularmente útil na utilização do processador para executar micro-instruções em paralelo ao acesso à memória, desde que essas não utilizem o dado acessado. O processador, portanto, não permanece ocioso durante esse tempo.

Exemplo:

1	LE	Iniciado o acesso
2	4 → CNTD	Carrega contador com 4
3	R1 + P → P, D	Soma os conteúdos dos registradores R1 da rede e P, armazenando o resultado em P. A micro-instrução 3 é executada, após a qual o controle entra no estado de espera até o dado lido estar no registrador D
4	D → Q	O dado em D pode agora ser manipulado transferindo ao registrador Q
	.	
	.	
	.	

A.2 CAMPO A



$(00)_2$ - NOP

$(01)_2$ - SRED

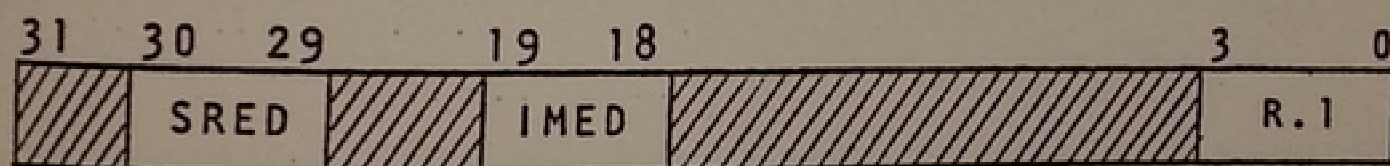
$(10)_2$ - I

$(11)_2$ - C

NOP NENHUMA OPERAÇÃO. Esta micro-ordem não seleciona nenhum registrador para a entrada A da ULA.

SRED SAÍDA DA REDE. Esta micro-ordem seleciona a saída da rede de registradores para a entrada A da ULA. Qualquer registrador pode ser selecionado desde que ele seja endereçado por micro-ordens adicionais nos campos MUXE e END.

Exemplo:



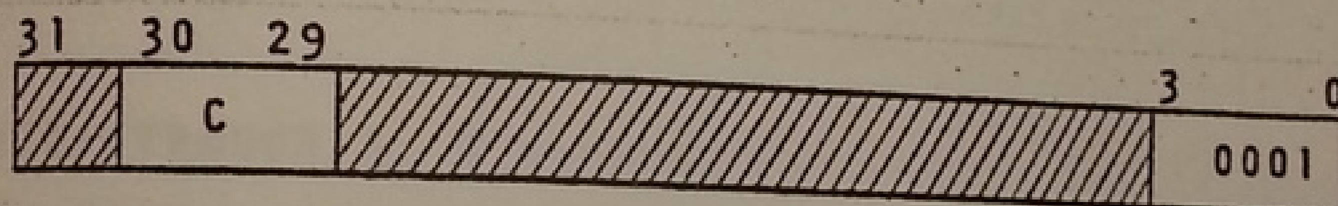
O registrador R.1 (corresponde ao registrador de trabalho SP1) é selecionado, e seu conteúdo terá acesso à entrada A da ULA.

I REGISTRADOR DE INSTRUÇÕES. Esta micro-ordem seleciona o registrador de instruções I para entrada A da ULA.

C REGISTRADORES ESPECIAIS. Esta micro-ordem seleciona o registrador de chaves do painel, vetor de interrupção VI, ou registrador de endereços E para a entrada A da ULA. Deve ser usado o campo END para especificar qual das opções será acessada.

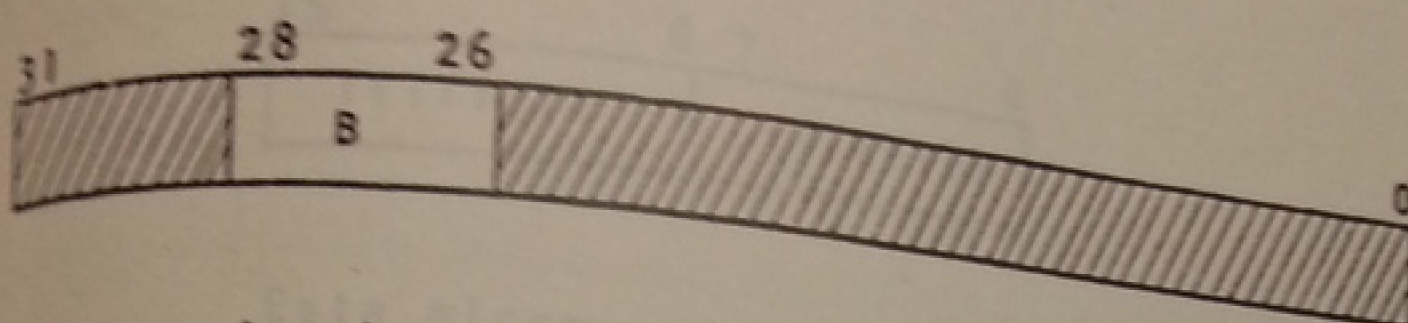
CAMPO END	CÓDIGO	ESPECIFICAÇÃO
EE	$(0001)_2$	Registrador E
PNL	$(0010)_2$	Registrador de Chaves do Painel
VI	$(1000)_2$	Vetor de Interrupção

Exemplo:



O registrador de endereços E é selecionado para a entrada A da ULA.

A.3 CAMPO B



$(000)_2$ ou $(100)_2$ - Q

$(001)_2$ ou $(101)_2$ - P

$(010)_2$ - DP

$(011)_2$ ou $(111)_2$ - S

$(110)_2$ - D

Q REGISTRADOR Q. O conteúdo do registrador Q é selecionado para a entrada B da ULA.

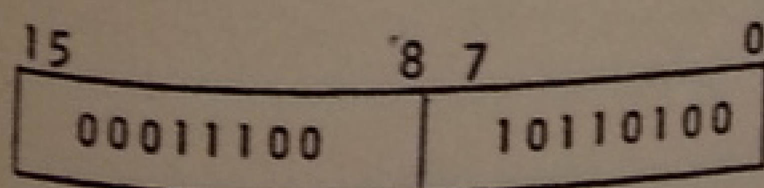
P REGISTRADOR P. O conteúdo do registrador P é selecionado para a entrada B da ULA.

DP REGISTRADOR D COM SINAL EXPANDIDO. Esta micro-orden transfere para a entrada B da ULA uma palavra composta pelos seguintes bytes:

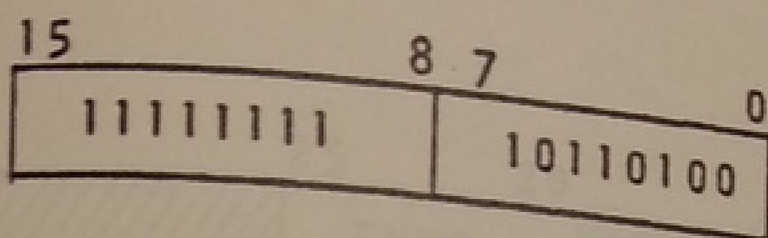
Byte menos significativo: igual ao byte direito do registrador D.

Byte mais significativo : contém em todos seus bits o bit 7 do registrador D.

Exemplo:
Se o registrador D contém a seguinte configuração de bits:



a palavra selecionada por DP será:

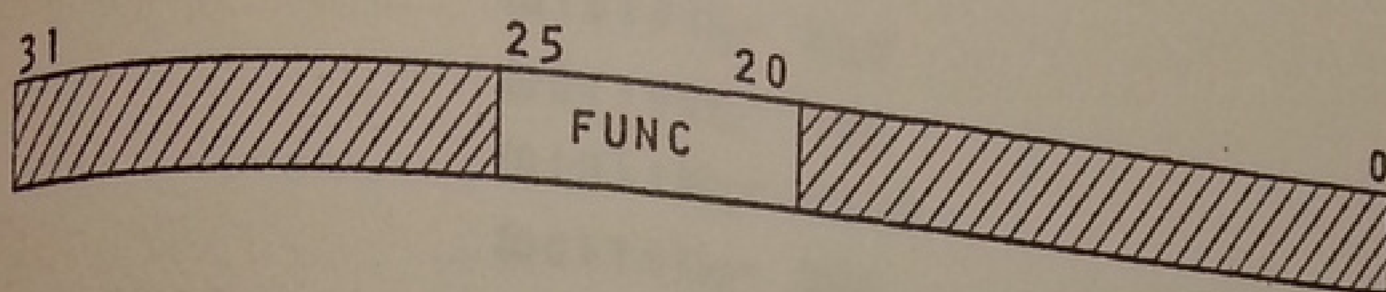


Esta micro-ordem é útil nas operações que envolvem bytes ou operandos de 8 bits. O sinal do byte direito é expandido para o byte esquerdo, e as operações lógicas e aritméticas são efetuadas normalmente como palavras de 16 bits.

S REGISTRADOR DE ESTADO. O registrador de estado S é selecionado para a entrada B da ULA.

D REGISTRADOR DE DADOS. O registrador D é selecionado para a entrada B da ULA.

A.4 CAMPO FUNC.



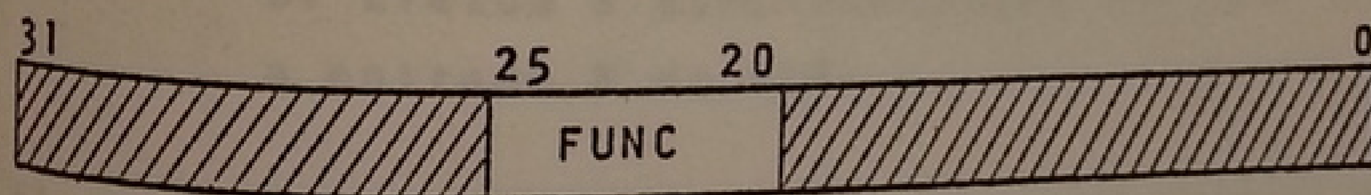
As micro-ordens desse campo estão divididas em três grupos:

- Micro-ordens de operações lógicas da ULA.
- Micro-ordens de operações aritméticas da ULA.
- Micro-ordens de operações de deslocamentos efetuados nos registradores P e Q.

Algumas atualizam os bits E, T, S, Z e P do registrador de estado e outras não.

As operações de deslocamento especificam apenas o tipo de deslocamento. O sentido (para a direita ou para a esquerda) é determinado por micro-ordens do campo D.

GRUPO 1 - MICRO-ORDENS DE OPERAÇÕES LÓGICAS DA ULA



$(111000)_2$ - A1
$(011011)_2$ - A2
$(110000)_2$ - A1S
$(010011)_2$ - A2S
$(001010)_2$ - BL
$(001000)_2$ - CMPA
$(001001)_2$ - CMPB

$(001011)_2$- EL
$(010101)_2$- ELS
$(001100)_2$- OUL
$(010110)_2$- OULS
$(001101)_2$- OUE
$(010111)_2$- OUES
$(010100)_2$- CP1S
$(110100)_2$- CP2S

A1, A2 — A. Efetua a passagem do dado selecionado para a entrada A da ULA sem alteração do mesmo.

A1S,A2S — A,ATUALIZANDO ESTADO. Efetua a mesma operação de A1 e A2. O registrador S é atualizado de acordo com o resultado da operação.

BL — B. Efetua a passagem do dado selecionado para a entrada B da ULA, sem alteração do mesmo.

CMPA — $\overline{A}$. Efetua a complementação do dado selecionado para a entrada A da ULA.

CMPB — $\overline{B}$. Efetua a complementação do dado selecionado para a entrada B da ULA.

EL — A Λ B. Efetua a operação "E" lógico entre os dados selecionados para as entradas A e B da ULA.

ELS — A Λ B ATUALIZANDO ESTADO. Efetua a mesma operação de EL. O registrador S é atualizado de acordo com o resultado da operação.

OUL — A V B. Efetua a operação "OU" lógico entre os dados selecionados para as entradas A e B da ULA.

OULS A V B ATUALIZANDO ESTADO. Efetua a mesma operação de OUL. O registrador S é atualizado de acordo com o resultado da operação.

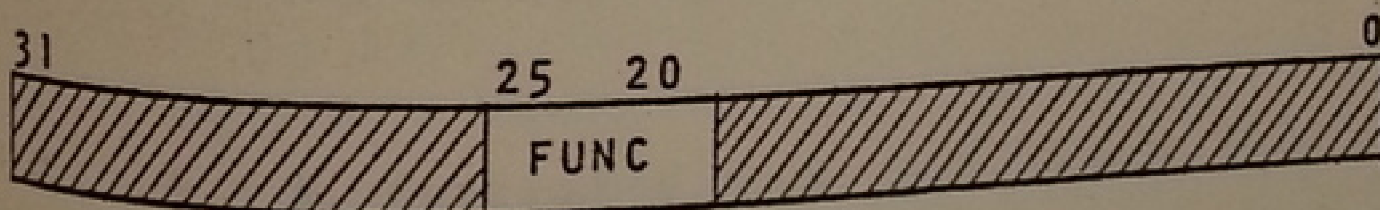
OUE A ⊕ B. Efetua a operação "OU" excludente lógico entre os dados selecionados para as entradas A e B da ULA.

OUES A ⊕ B ATUALIZANDO ESTADO. Efetua a mesma operação de OUE. O registrador S é atualizado de acordo com o resultado da operação.

CP1S A V $\overline{B}$ ATUALIZANDO ESTADO. Efetua a operação A "ou" complemento de 1 de B com os dados selecionados para as entradas A e B da ULA. O registragor S é atualizado de acordo com o resultado da operação.

CP2S A V $\overline{B}$ + 1. Efetua a operação A "ou" complemento de 1 de B mais 1 com os dados selecionados para as entradas A e B da ULA. O registrador S é atualizado de acordo com o resultado da operação.

GRUPO 2 - MICRO-ODENS DE OPERAÇÕES ARITMÉTICAS DA ULA.



$(011010)_2$- SOM
$(010010)_2$- SOMS
$(111001)_2$- SUB
$(110001)_2$- SUBS
$(011001)_2$- DIM
$(010001)_2$- DIMS

$(111010)_2$- ADI
$(110010)_2$- ADS
$(111011)_2$- ICR
$(110011)_2$- ICRS
$(011000)_2$- DCR
$(010000)_2$- DCRS
$(110111)_2$- TIPS

SOM A + B. Soma os dados selecionados para as entradas A e B da ULA.

SOMS A + B ATUALIZANDO ESTADO. Efetua a mesma operação do SOM. O registrador S é atualizado de acordo com o resultado da operação.

SUB A - B. Subtrai o dado selecionado para a entrada B do dado selecionado para a entrada A da ULA.

SUBS A - B ATUALIZANDO ESTADO. Efetua a mesma operação de SUB. O registrador S é atualizado de acordo com o resultado da operação.

DIM A - B. Subtrai o dado selecionado para a entrada B do dado selecionado para a entrada A da ULA, decrementando o resultado de 1.

DIMS A - B - 1 ATUALIZANDO ESTADO. Efetua a mesma operação de DIM. O registrador S é atualizado de acordo com o resultado da operação.

ADI A + B + 1. Soma os dados selecionados para as entradas A e B da ULA, incrementando o resultado de 1.

ADS A + B + 1 ATUALIZANDO ESTADO. Efetua a mesma operação de ADI. O registrador S é atualizado de acordo com o resultado da operação.

ICR A + 1. Incrementa de 1 o dado selecionado para a entrada A da ULA.

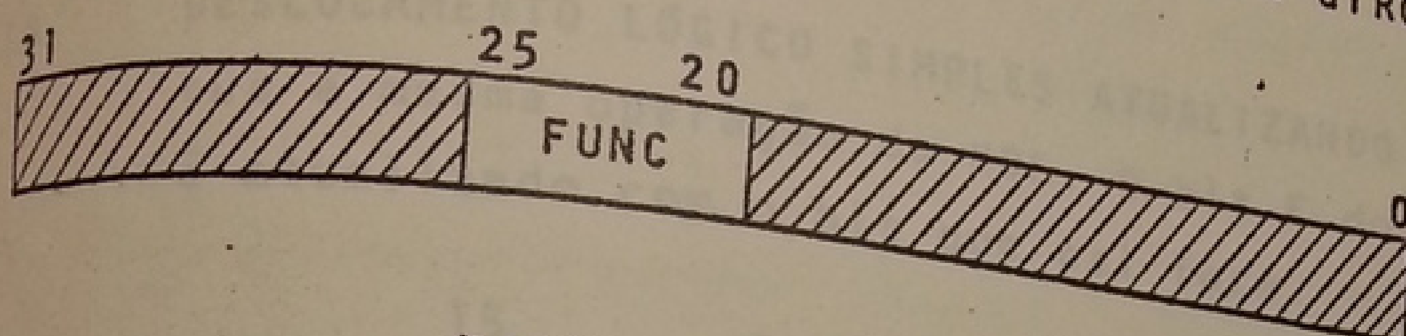
ICRS A + 1 ATUALIZANDO ESTADO. Efetua a mesma operação de ICR. O registrador S é atualizado de acordo com o resultado da operação.

DCR A - 1. Decrementa de 1 o dado selecionado para a entrada A da ULA.

DCRS A - 1 ATUALIZANDO ESTADO. Efetua a mesma operação de DCR. O registrador S é atualizado de acordo com o resultado da operação.

TIPS É uma micro-ordem genérica que efetua a operação lógica aritmética ou de deslocamento especificada pelos bits <8 : 10> das instruções aritméticas e lógicas imediatas, curtas, de registradores e de bytes; e operações de deslocamentos e giros das instruções de deslocamentos e giros.

GRUPO 3 - MICRO-ORDENS DE DESLOCAMENTOS E GIROS

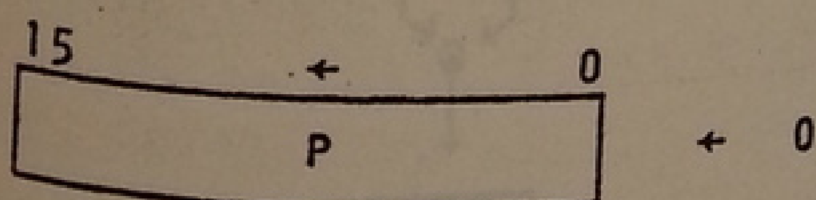


$(011100)_2$ - MOS
$(000000)_2$ - LS
$(000001)_2$ - AS
$(011101)_2$ - GIRS
$(000010)_2$ - GS
$(000011)_2$ - GES
$(000100)_2$ - LD
$(000101)_2$ - AD
$(000110)_2$ - GD
$(011110)_2$ - GIRD
$(000111)_2$ - GED
$(001110)_2$ - DES
$(001111)_2$ - DED

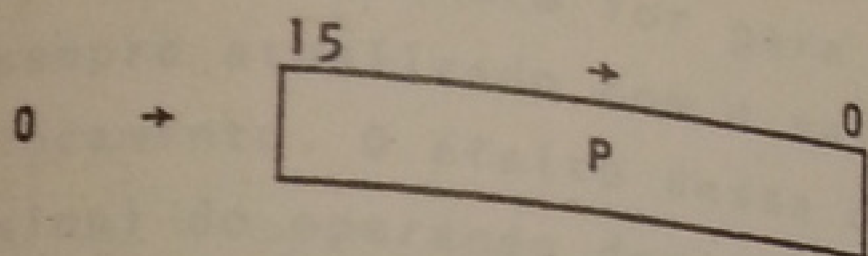
Estas micro-ordens devem ser usadas com as do campo D que determinam o sentido dos deslocamentos.

MOS

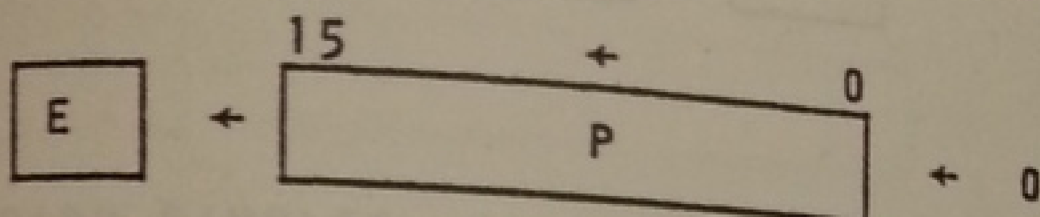
DESLOCAMENTO LÓGICO SIMPLES. Desloca o dado no registrador P de uma posição, entrando 0 em P <0>se o deslocamento for para a esquerda,

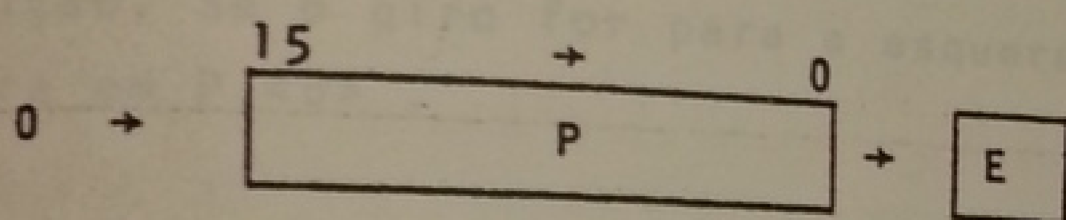


ou 0 em P < 15 > se o deslocamento for para a direita.

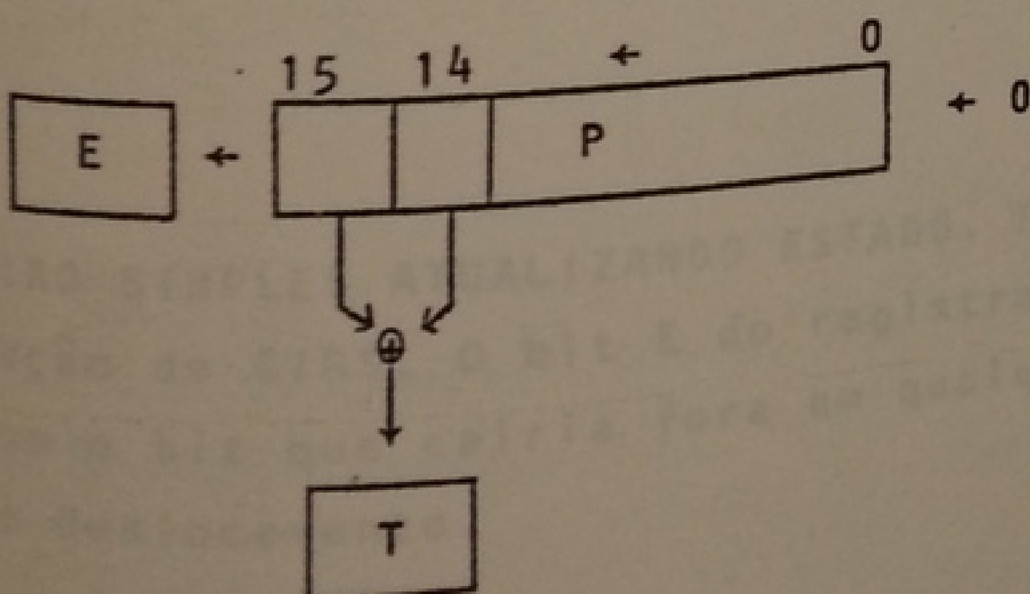


LS DESLOCAMENTO LÓGICO SIMPLES ATUALIZANDO ESTADO. Efetua a mesma operação de MOS. O bit E do registrador S é atualizado com o bit que cai fora.

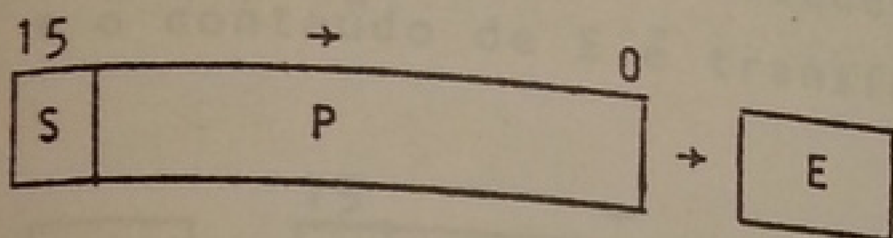




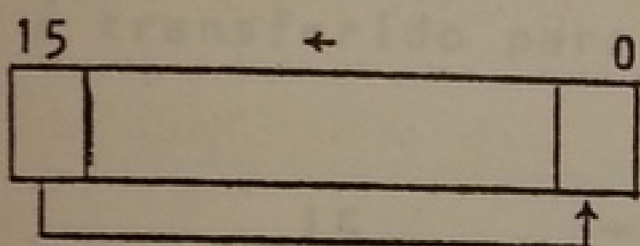
AS DESLOCAMENTO ARITMÉTICO SIMPLES ATUALIZANDO ESTADO. Desloca o dado no registrador P de uma posição. Se o deslocamento for para a esquerda, entra 0 em P <0>. O bit E do registrador S é atualizado com o bit que cai fora, e o bit T é atualizado segundo o resultado de um "OU" excludente dos bits P <15> e P <14> .



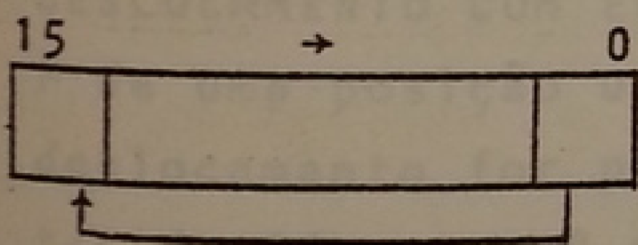
Se o deslocamento for para a direita, o bit P <15> é sempre atualizado com o valor original antes do deslocamento. O efeito dessa operação é de conservar o sinal do operando deslocado. O bit que cai fora pela direita é armazenado no bit E do registrador S.



GIRS GIRO SIMPLES. Gira o dado no registrador P de uma posição. Se o giro for para a esquerda, o bit P <15> entra em P <0> .

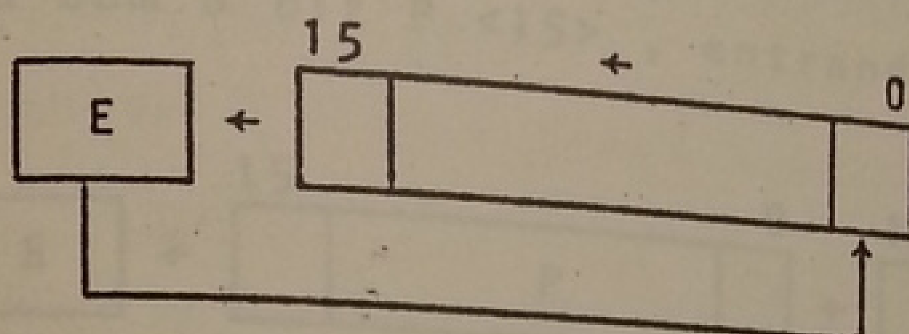


Se o giro for para a direita, P <0> entra no bit P <15> .

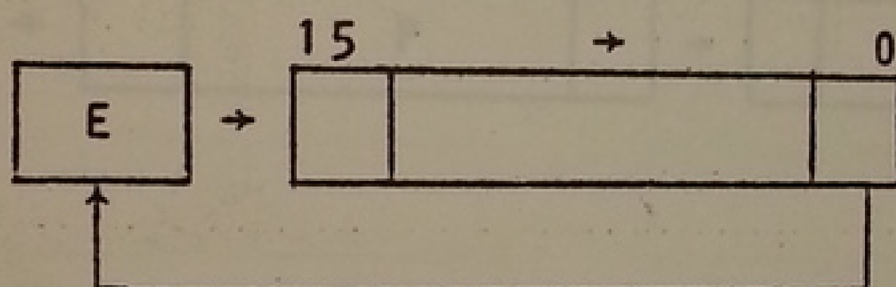


GS GIRO SIMPLES ATUALIZANDO ESTADO. Efetua a mesma operação de GIRS. O bit E do registrador S é atualizado com o bit que cairia fora em qualquer um dos sentidos de deslocamento.

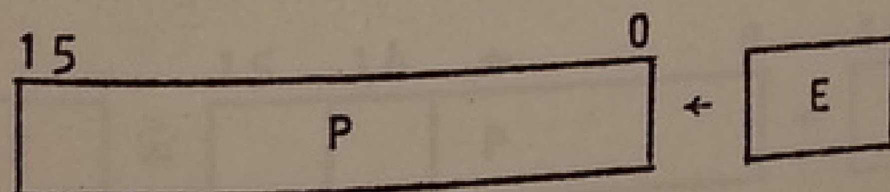
GES GIRO SIMPLES COM EXTENSÃO ATUALIZANDO ESTADO: Gira o dado no registrador P incluindo o bit de extensão E . Se o giro for para a esquerda o bit P <15> entra em E, e o conteúdo de E é transferido para P <0> .



Se o giro for para a direita, o bit P <0> entra no bit E do registrador S, ao passo que o conteúdo de E é transferido para P <15> .

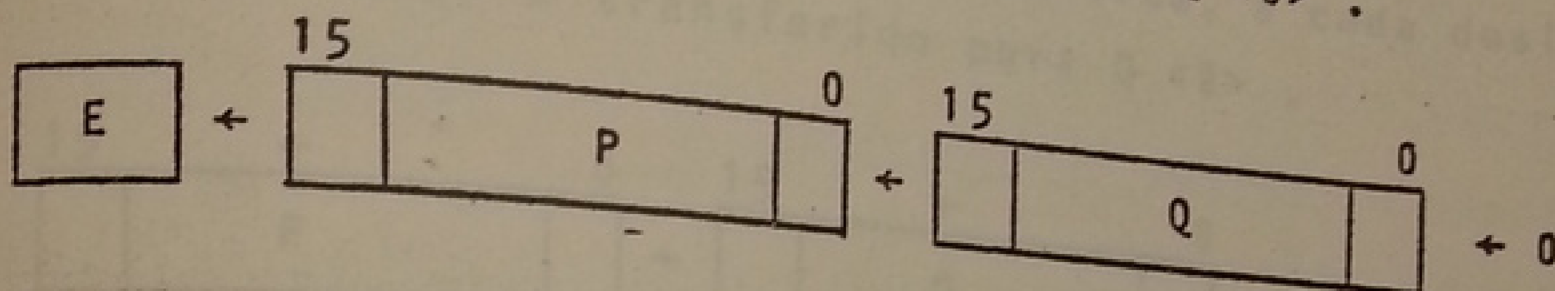


DES DESLOCAMENTO COM EXTENSÃO SIMPLES. Desloca o dado em P de uma posição utilizando o bit E de extensão. Se o deslocamento for para a esquerda, o conteúdo de E é tranferido para P <0>

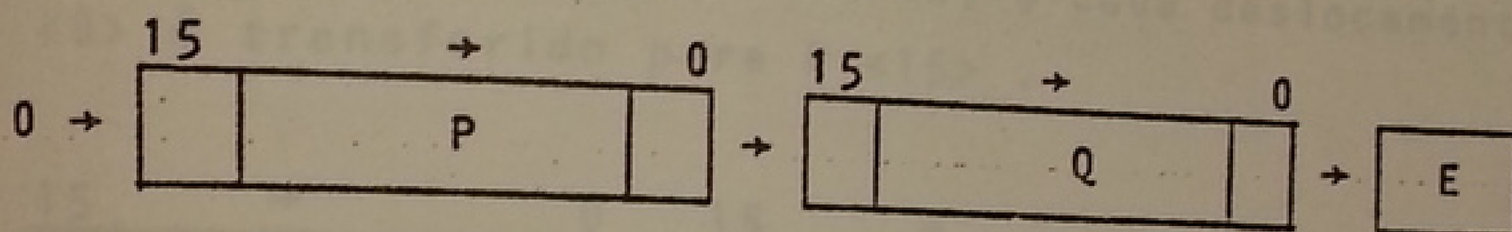


Se o deslocamento for para a esquerda, o conteúdo de E é transferido para P <15>.

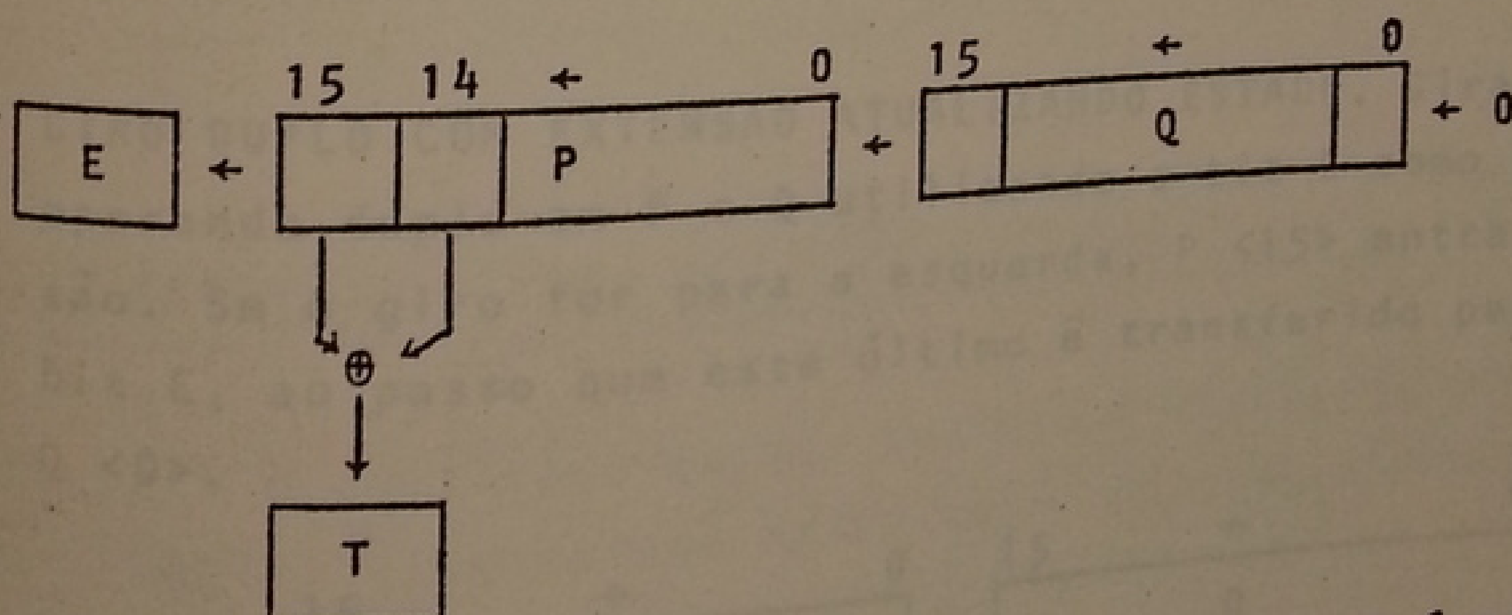
LD DESLOCAMENTO LÓGICO DUPLO ATUALIZANDO ESTADO. Desloca o operando duplo cuja palavra mais significa está no registrador P e a menos significativa, no registrador Q, de uma posição. Se o sentido de deslocamento for para a esquerda, o bit E do registrador S é atualizado com o bit P <15> , entrando 0 em Q <0> .



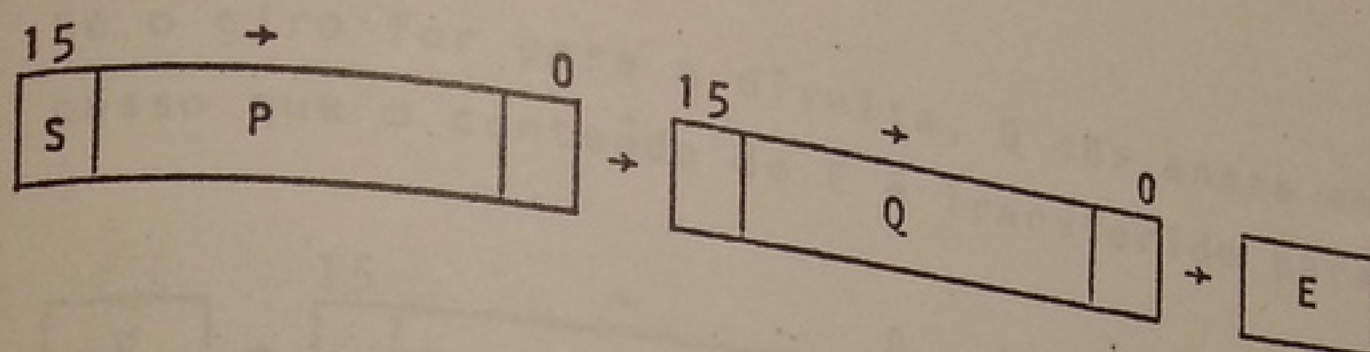
Se o sentido de deslocamento for para a direita, o bit E é atualizado com o bit Q <0> , entrando 0 em P <15>



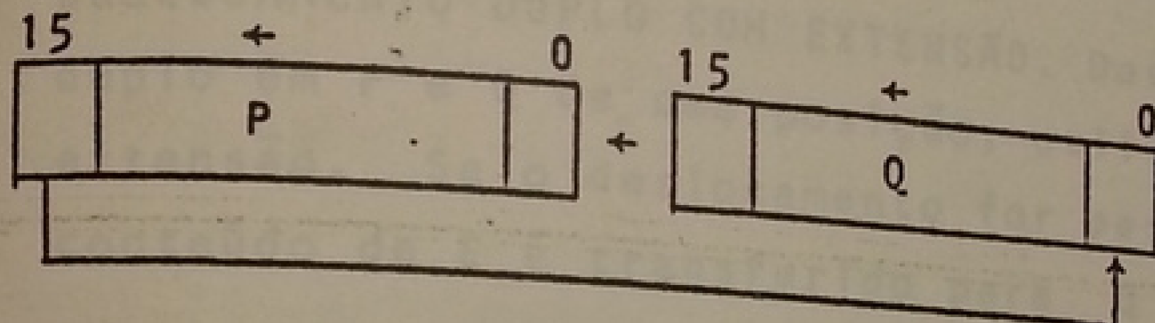
AD DESLOCAMENTO ARITMÉTICO DUPLO ATUALIZANDO ESTADO. Desloca o operando duplo em P e Q de uma posição. Se o deslocamento for para a esquerda, o bit E é atualizado com o bit que cai fora, e o bit T é atualizado segundo um "OU" excludente de P <14> e P <15> . Entra 0 no bit Q <0> .



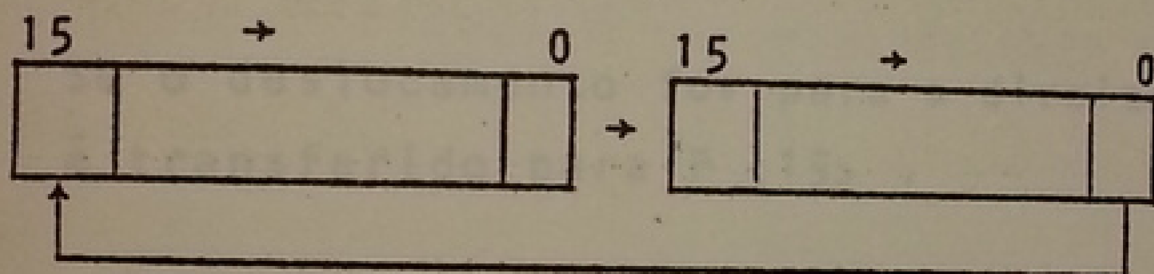
Se o deslocamento for para a direita, o bit E é atualizado com o bit que cai fora, e P <15> é sempre atualizado com o valor original antes do deslocamento. O efeito dessa operação é de conservar o sinal de operando deslocado.



GIRD GIRO DUPLO. Gira o operando duplo em P e Q de uma posição. Se o giro for para a esquerda, a cada deslocamento, P <15> é transferido para Q <0> .

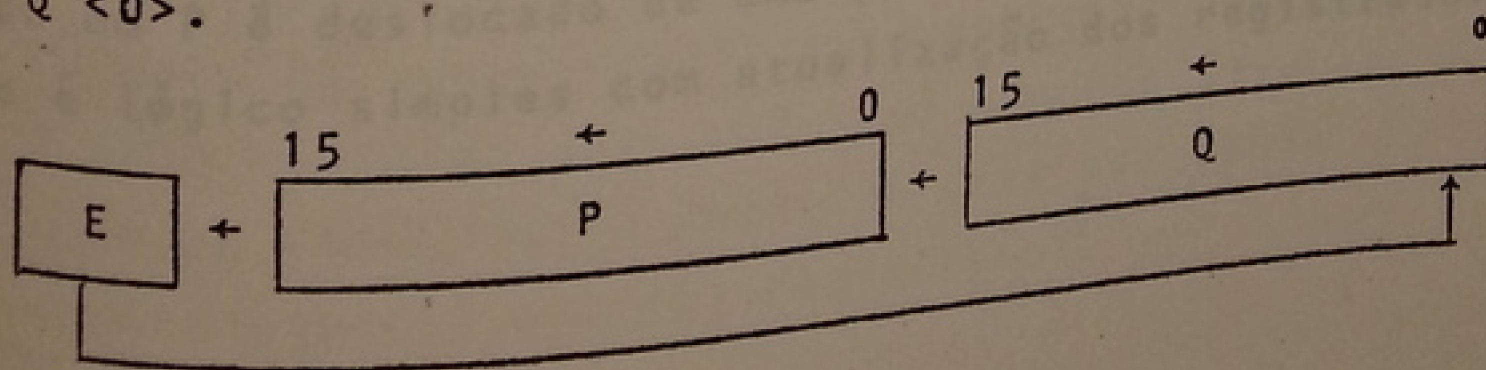


Se o giro for para a direita, a cada deslocamento Q <0> é transferido para P <15> .

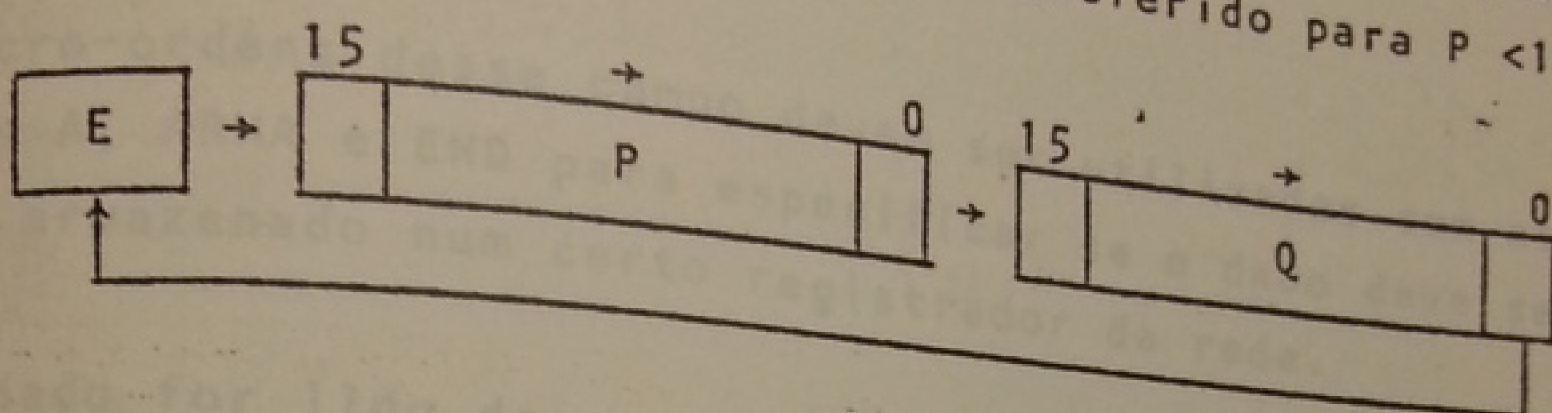


GD GIRO DUPLO ATUALIZANDO ESTADO. Efetua a mesma operação de GIRD. O bit E do registrador S é atualizado com o bit que cairia fora.

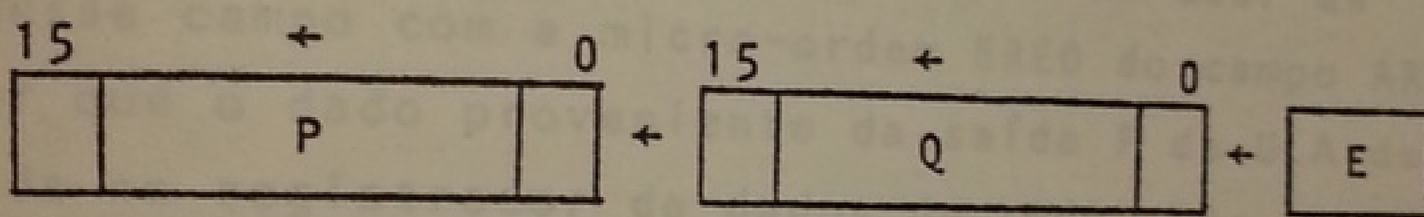
GED GIRO DUPLO COM EXTENSÃO ATUALIZANDO ESTADO. Gira o operando duplo em P e Q utilizando o bit E como extensão. Se o giro for para a esquerda, P <15> entra no bit E, ao passo que este último é transferido para Q <0>.



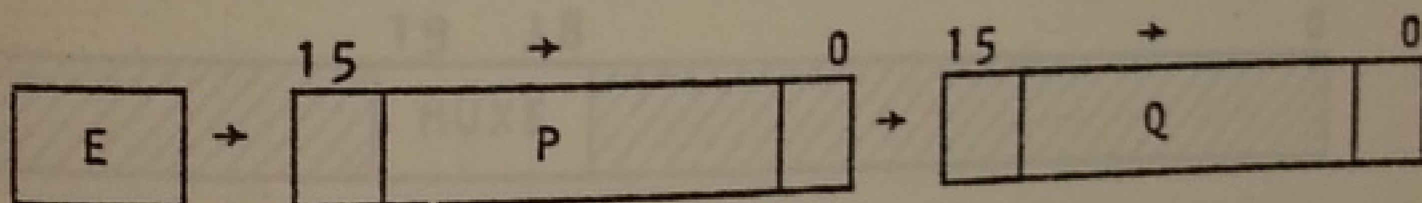
Se o giro for para a direita, Q <0> entra em E, ao passo que o conteúdo de E é transferido para P <15> .



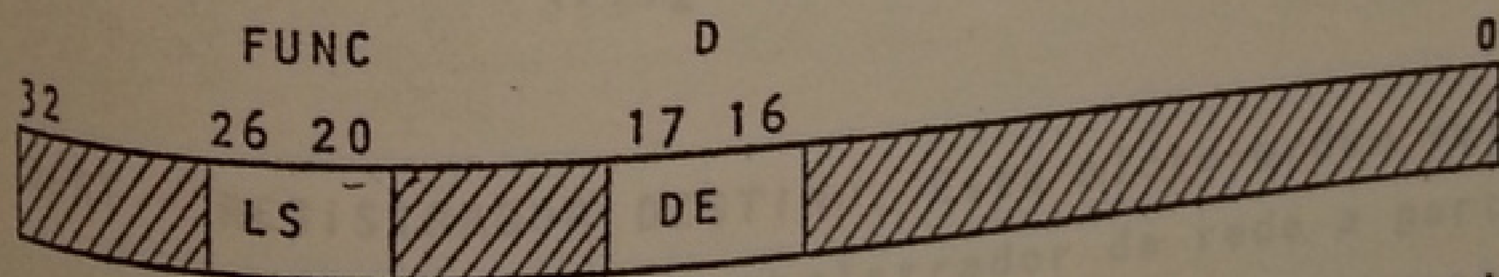
DED DESLOCAMENTO DUPLO COM EXTENSÃO. Desloca o operando duplo em P e Q de uma posição, utilizando o bit E de extensão. Se o deslocamento for para a esquerda, o conteúdo de E é transferido para Q <0> .



Se o deslocamento for para a direita, o conteúdo de E é transferido para P <15> .



Exemplo de um deslocamento, utilizando os campos FUNC e D :



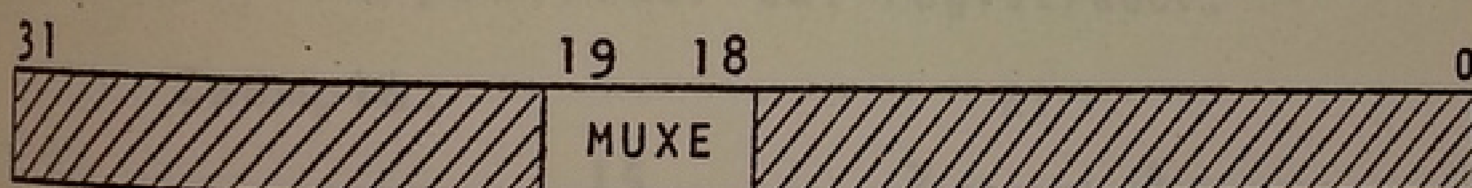
O dado em P é deslocado de uma posição à esquerda. O deslocamento é lógico simples com atualização dos registrador S.

A.5 CAMPO MUXE

As micro-ordens desse campo devem ser utilizadas com as dos campos A, ARMA e END para especificar se o dado deve ser lido ou armazenado num certo registrador da rede.

Se o dado for lido da rede, reve-se usar as micro-ordens desse campo com a micro-ordem SRED do campo A para indicar que a saída da rede foi selecionada para a entrada A da ULA; no campo END pode vir especificado o registrador a ser lido.

Se o dado for armazenado na rede, deve-se usar as micro-ordens desse campo com a micro-ordem ERED do campo ARMA para indicar que o dado proveniente da saída F da ULA deve ser armazenado no registrador de dados da rede, DR; no campo END pode vir especificado o registrador onde será armazenado o dado.

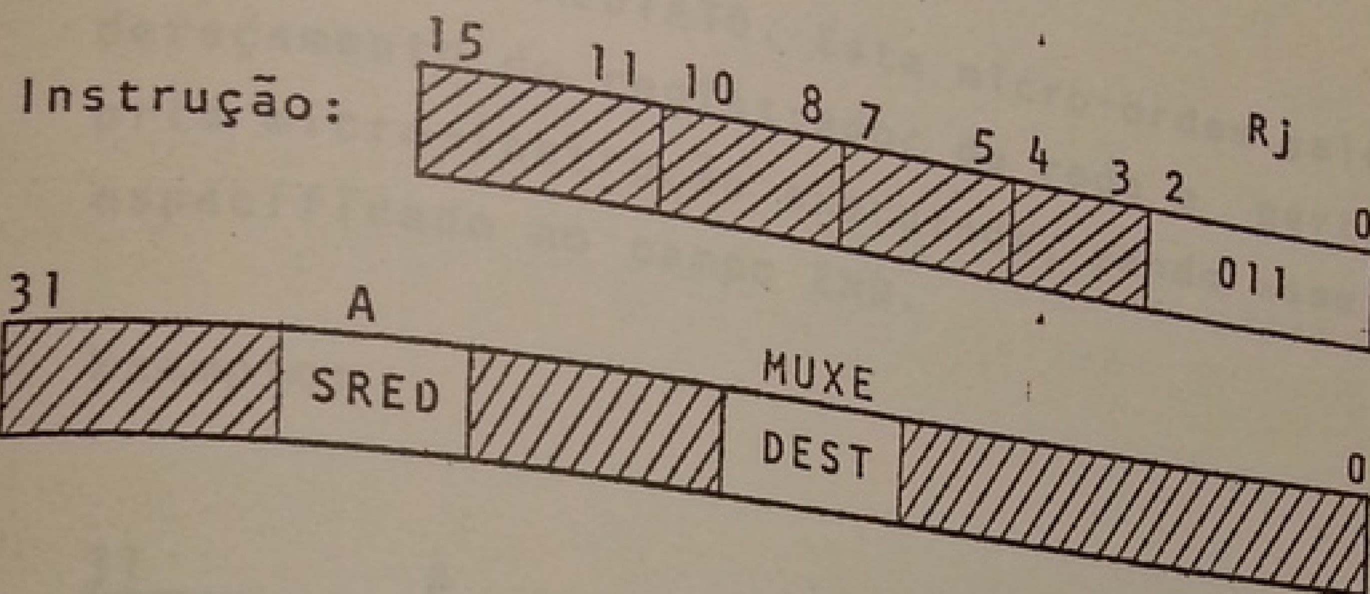


$(00)_2$ - DEST
$(01)_2$ - ORGM
$(10)_2$ - IMED
$(11)_2$ - CNT

DEST - REGISTRADOR DESTINO. Esta micro-ordem seleciona o endereçamento do registrador da rede a partir dos bits <0 : 2> da instrução de máquina, correspondente ao campo Rj. Nesse campo está especificado o registrador cujo conteúdo deve ser lido ou onde deve ser armazenado um dado. Nesse caso não deve ser usado o campo END para especificar tal registrador.

Exemplo:

Instrução:

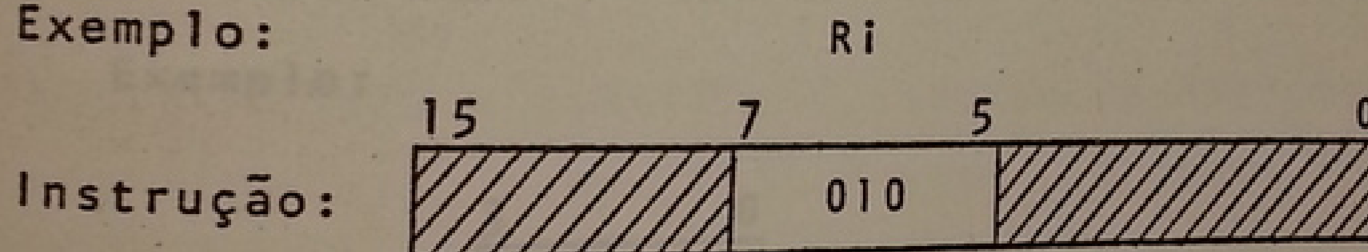


O registrador R3 é selecionado para a entrada A da ULA.

ORGM REGISTRADOR ORIGEM. Esta micro-ordem seleciona o endereçamento do registrador da rede a partir dos bits <5 : 7> da instrução de máquina, correspondente ao campo Ri. Nesse campo está especificado o registrador cujo conteúdo deve ser lido ou onde deve ser armazenado um dado. Nesse caso não deve ser usado o campo END para especificar tal registrador.

Exemplo:

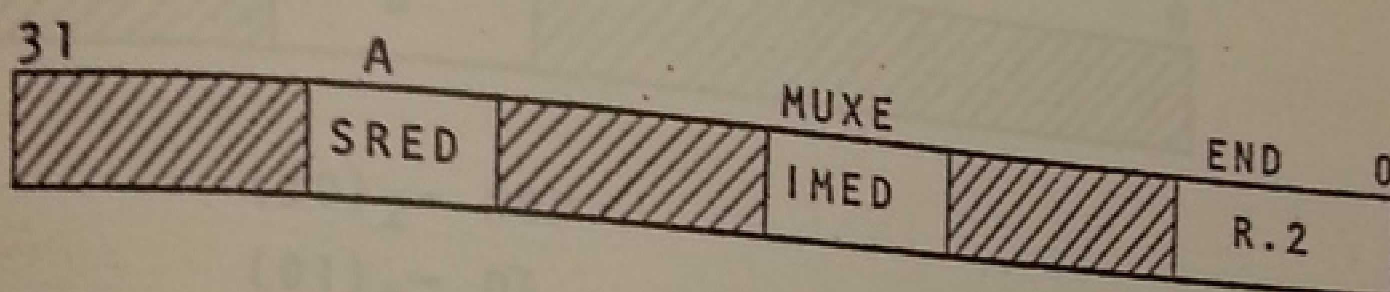
Instrução:



O registrador R2 é selecionado para a entrada A da ULA.

IMED REGISTRADOR IMEDIATO. Esta micro-ordem seleciona o endereçamento do registrador da rede a partir da própria micro-instrução executada, sendo esse registrador especificado no campo END.

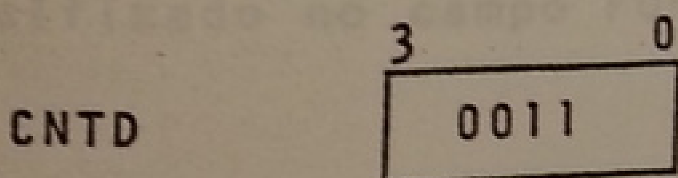
Exemplo:

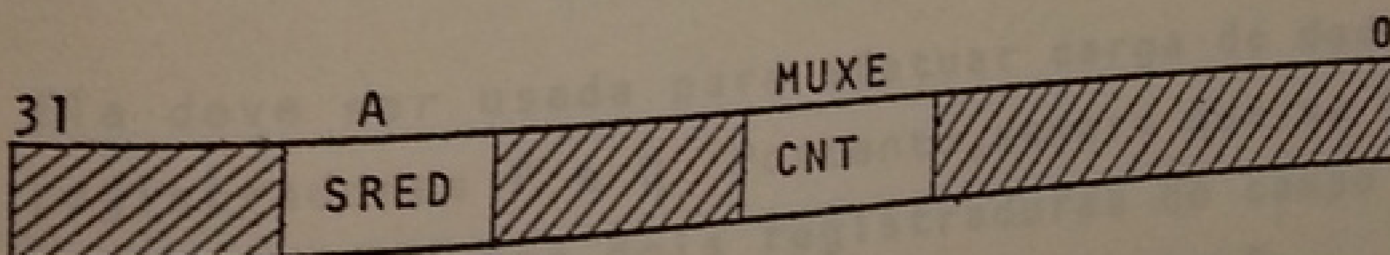


O conteúdo do registrador R.2 da rede, correspondente ao registrador de trabalho SP2 é lido.

CNT ENDEREÇAMENTO PELO CONTADOR. Esta micro-ordem permite o endereçamento sequencial direto dos registradores da rede através do contador de 4 bits, CNTD, sem a necessidade desses serem especificados no campo END.

Exemplo:

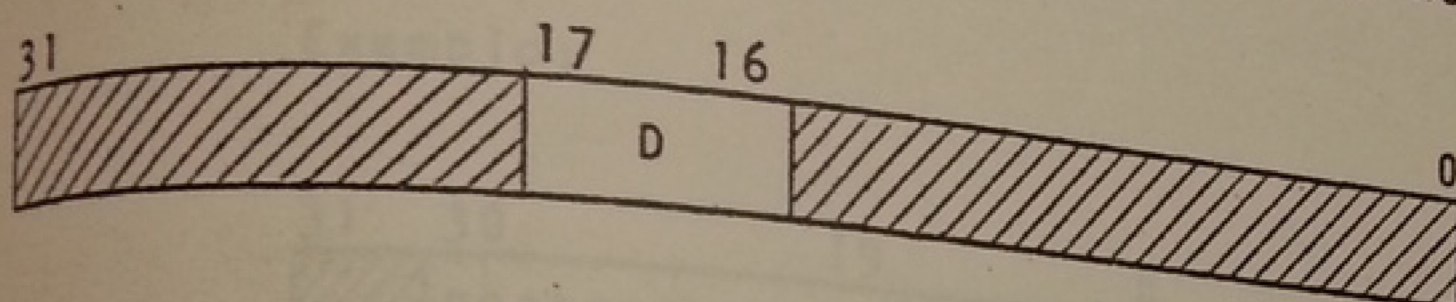




O conteúdo do registrador R.3 correspondente ao registrador de trabalho SP3 é lido.

A.6 CAMPO D

As micro-ordens desse campo, exceto à última (CAR) se referem ao sentido do deslocamento especificado no campo FUNC.



$(00)_2$ - NOP

$(01)_2$ - DE

$(10)_2$ - DD

$(11)_2$ - CAR

NOP NENHUMA OPERAÇÃO. Não é executada nenhuma operação.

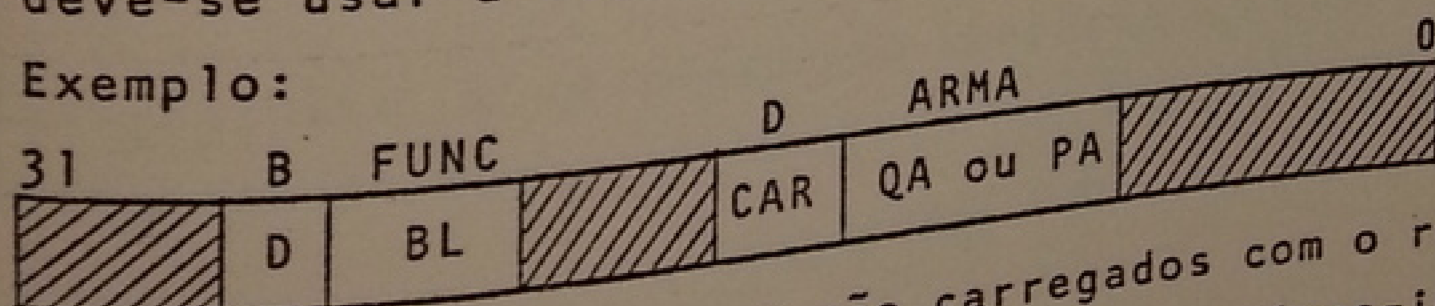
DE DESLOCAMENTO A ESQUERDA. O sentido do deslocamento especificado no campo FUNC é para a esquerda.

DD DESLOCAMENTO A DIREITA. O sentido do deslocamento especificado no campo FUNC é para a direita.

CAR CARGA. Esta micro-ordem tem duas funções:

1) Ela deve ser usada para efetuar carga de dados nos registradores P e Q. Portanto, sempre que se faz referência a esses dois registradores no campo ARMA, deve-se usar a micro-odem CAR no campo D.

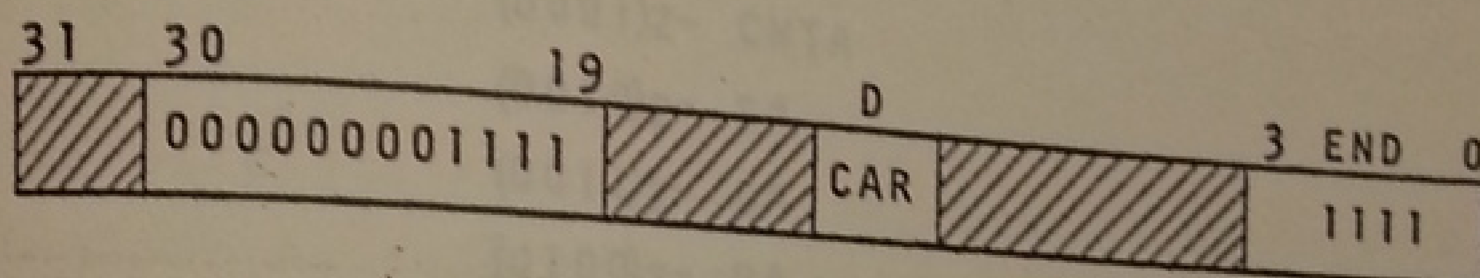
Exemplo:



Os registradores Q ou P são carregados com o resultado da operação efetuada na ULA nessa micro-instrução.

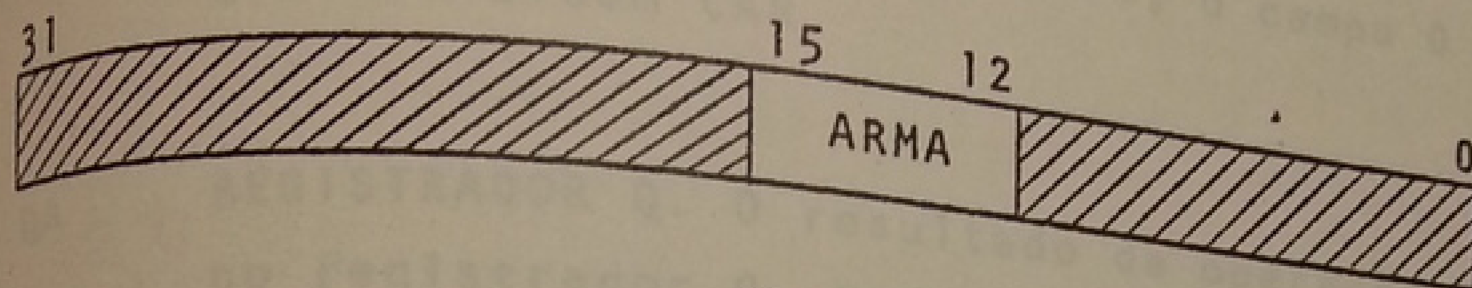
2) A micro-ordem CAR também é usada para carregar o registrador P com uma constante. Nesse caso, não deve constar nenhuma micro-ordem no campo ARMA. A constante vem especificada nos bits <0 : 3> e <19 : 30> da micro-instrução.

Exemplo:



A constante /00FF é carregada no registrador P.

A.7 CAMPO ARMA



$(0000)_2$ - NOP
$(0001)_2$ - CNTA
$(0010)_2$ - SA
$(0011)_2$ - DA
$(0100)_2$ - PA
$(0101)_2$ - QA
$(0110)_2$ - ERED
$(0111)_2$ - IA
$(1000)_2$ - EA
$(1001)_2$ - RELD
$(1010)_2$ - PEA
$(1011)_2$ - RELF

NOP NENHUMA OPERAÇÃO. O resultado da operação efetuada nessa micro-instrução não é armazenada em nenhum registrador.

CNTA CONTADOR. O resultado da operação tem seus 4 bits menos significativos armazenados no contador CNTD.

SA REGISTRADOR DE ESTADO. O resultado da operação é armazenado no registrador S.

DA REGISTRADOR D. O resultado da operação é armazenado no registrador D.

PA REGISTRADOR P. O resultado da operação é armazenado no registrador P. Nesse caso, o campo D deve conter a micro-ordem CAR.

QA REGISTRADOR Q. O resultado da operação é armazenado no registrador Q. Nesse caso, o campo D deve conter a micro-ordem CAR.

ERED REDE DE REGISTRADORES. O resultado da operação é armazenado na rede de registradores. Nesse caso, deve-se usar uma das micro-ordens do campo MUXE, e se necessário, usar o campo END para especificar o registrador.

Exemplo:

	B	FUNC	MUXE		ARMA		END
	D	BL	IMED		ERED		R.3

O conteúdo do registrador D é armazenado no registrador R.3 da rede (correspondente ao registrador de trabalho SP3).

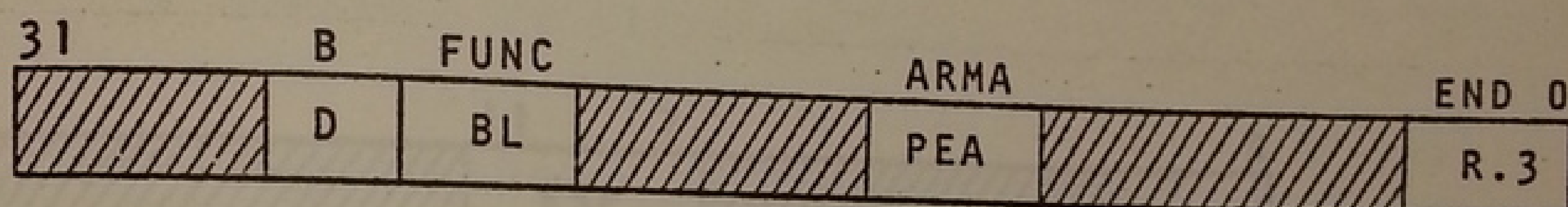
IA REGISTRADOR DE INSTRUÇÕES. O resultado da operação é armazenado no registrador I.

EA REGISTRADOR DE ENDEREÇOS. O resultado da operação é armazenado no registrador E.

RELD REGISTRADOR DE DADOS DO RELÓGIO INTERNO. O resultado da operação é armazenado no registrador de dados do Relógio Interno.

PEA BYTE DIREITO DO REGISTRADOR. O resultado da operação tem seu byte direito armazenado no byte direito de um dos registradores da rede especificado no campo END, sem alteração do byte esquerdo do mesmo registrador.

Exemplo:



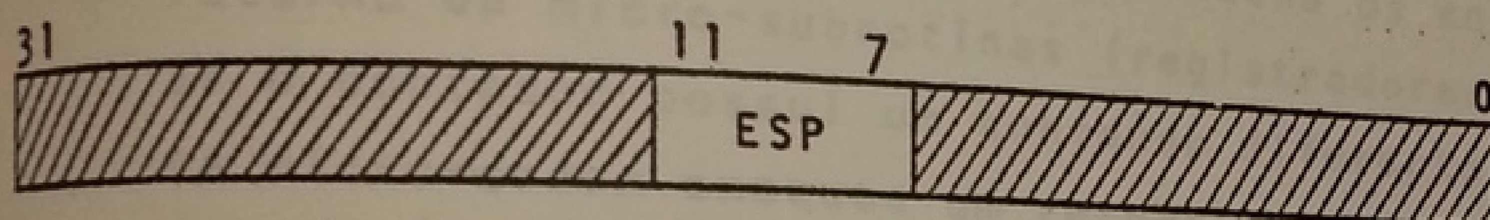
O byte direito do registrador D é armazenado no byte direito do registrador R.3 (correspondente ao registrador de trabalho SP3).

RELF REGISTRADOR DE FUNÇÃO DO RELÓGIO INTERNO. O resultado da operação é armazenado no registrador de função do Relógio Interno.

A.8 CAMPO ESP

Este é um campo especial que contém as micro-ordens que geram controle sobre a seleção do registrador EC ou sobre outros blocos do processador ou da via comum.

Algumas das micro-ordens usam o campo COND(micro-ordens condicionais), outras usam os bits <19 : 30> (pulo e pulo para micro-subrotina) para endereçamento da memória de controle , e ainda há as que usam o campo END para geração de constantes de 4 bits.



$(00000)_2$ - NOP
$(00001)_2$ - FIM
$(00010)_2$ - FIMC
$(00011)_2$ - SEBU
$(00100)_2$ - PLAC
$(00110)_2$ - PUGC
$(00111)_2$ - FOR1
$(01000)_2$ - FOR2
$(01001)_2$ - PAIN
$(01010)_2$ - FUNC
$(01011)_2$ - SALC
$(01100)_2$ - DCSC
$(01101)_2$ - SEVI
$(01110)_2$ - REPC
$(10000)_2$ - LE
$(10001)_2$ - RVC
$(10100)_2$ - CNTD
$(10101)_2$ - MODS
$(10111)_2$ - ESC
$(11000)_2$ - RECI

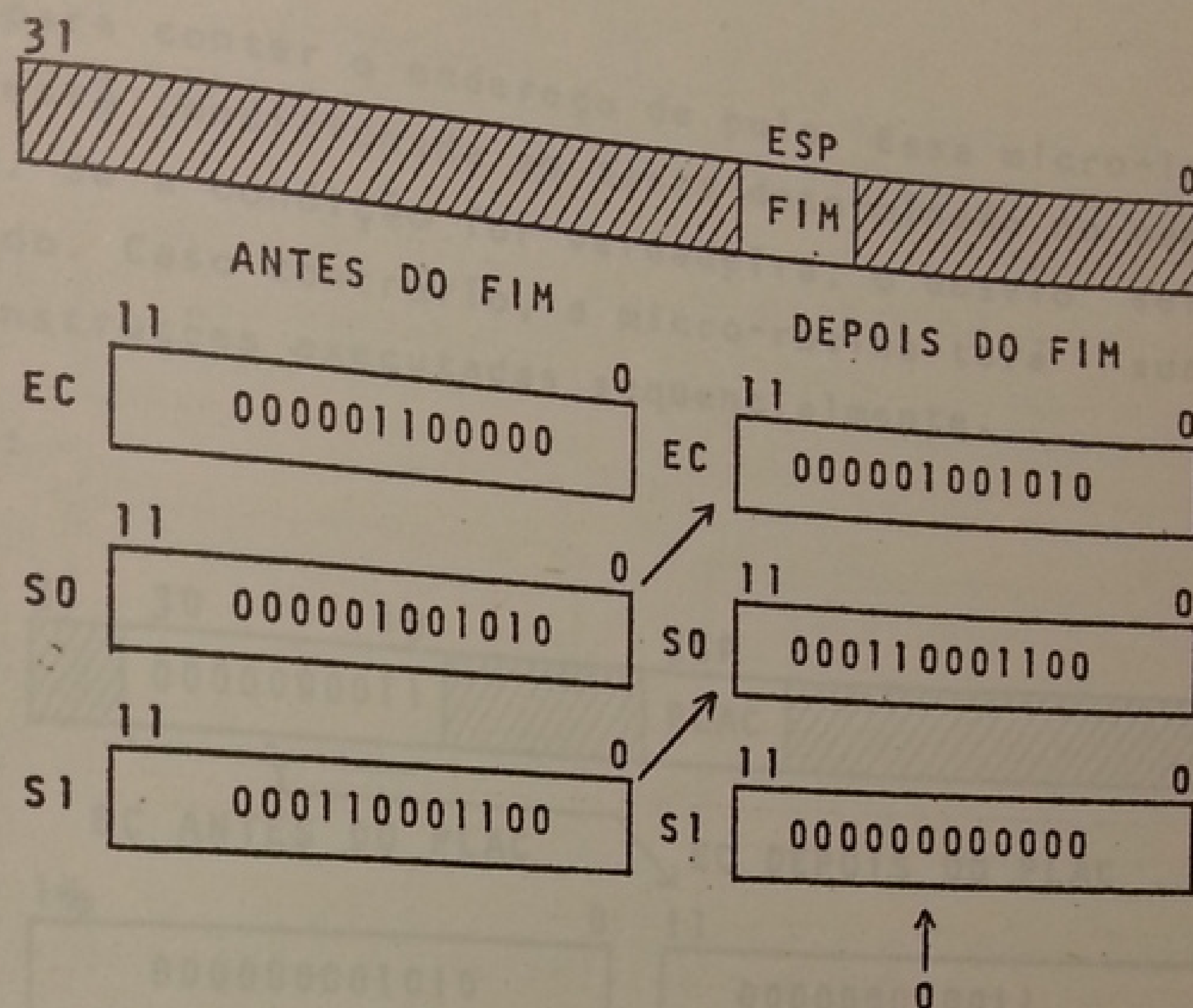
$(11001)_2$ - LINT
$(11010)_2$ - PERI
$(11011)_2$ - EEX
$(11100)_2$ - SETE
$(11101)_2$ - PAR

NOP NENHUMA OPERAÇÃO. Não é executada nenhuma operação.

FIM FINALIZAÇÃO DE MICRO-PROGRAMA. Esta micro-ordem marca o término de execução de uma micro-rotina. Ela age sobre o contador da pilha que armazena os endereços de retorno de micro-subrotinas (registradores S0, S1).
Se o contador possui um conteúdo diferente de zero esta micro-ordem executa um retorno de micro-subrotina: o conteúdo de S0 é transferido para EC e o conteúdo de S1 para S0, decrementando o contador de micro-subrotinas.
Se esse contador contém zero, esta micro-ordem determina o fim da fase de execução da instrução de máquina, passando para uma nova fase de busca da nova instrução, carregando no registrador de endereço da memória de controle (EC) o endereço da micro-rotina de busca (endereço /0000).

Caso essa micro-ordem determina o fim da fase de execução, ela poderá detetar um pedido de interrupção externo. Se isso acontecer, será feito um desvio automático para a micro-rotina tratadora dessa interrupção.

Exemplo:



FIMC FINALIZAÇÃO CONDICIONAL DE MICRO-PROGRAMA. Esta micro-instrução efetua a mesma operação da anterior, condicionada ao teste determinado no campo COND. Se a condição foi verdadeira, a micro-rotina é finalizada. Caso contrário, sua execução terá sequência normal.

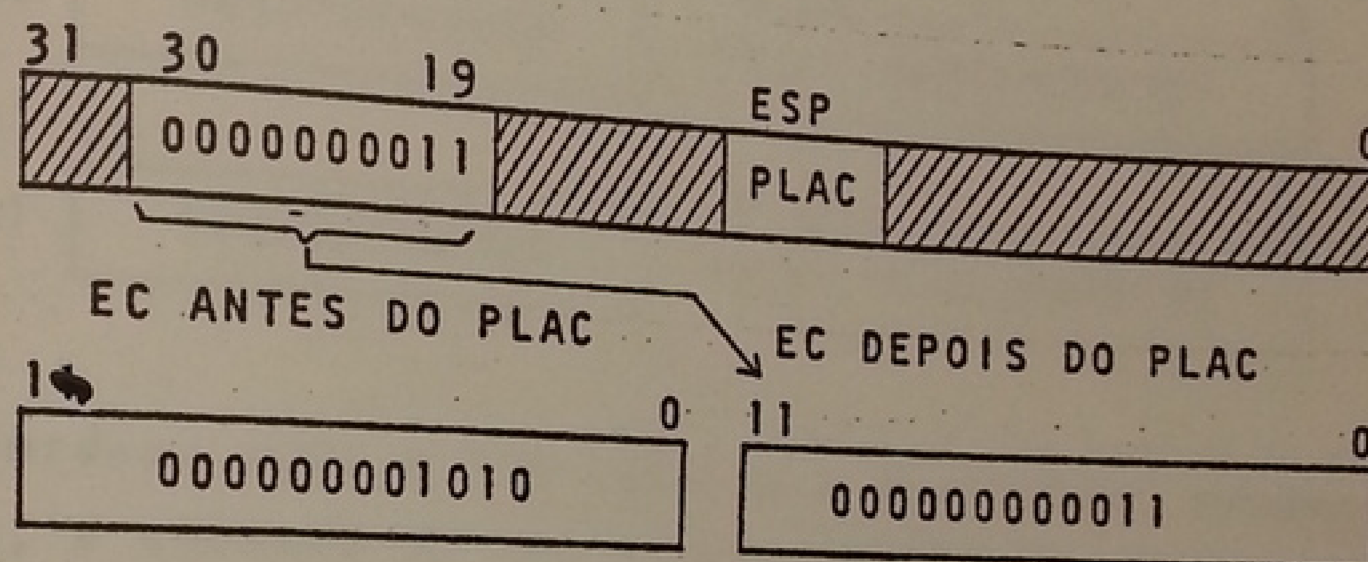
SEBU ACIONA FASE DE BUSCA. Esta micro-ordem é usada no início da micro-rotina de busca, e sua função é a de iniciar a fase de busca propriamente dita da nova instrução.

PLAC PULO CONDICIONAL. Essa micro-ordem permite a seleção dos bits <19 : 30> do registrador DC para EC, endereçando uma nova micro-instrução, quebrando a sequência normal de endereçamento.

Notar que quando for colocada esta micro-instrução no campo ESP, não se pode especificar nenhuma operação com a ULA, pois os campos A, B e FUNC estarão sendo

usados para conter o endereço de pulo. Essa micro-instrução está condicionada ao teste determinado no campo COND. Se a condição for verdadeira, o desvio será executado. Caso contrário, a micro-rotina terá suas micro-instruções executadas sequencialmente.

Exemplo:

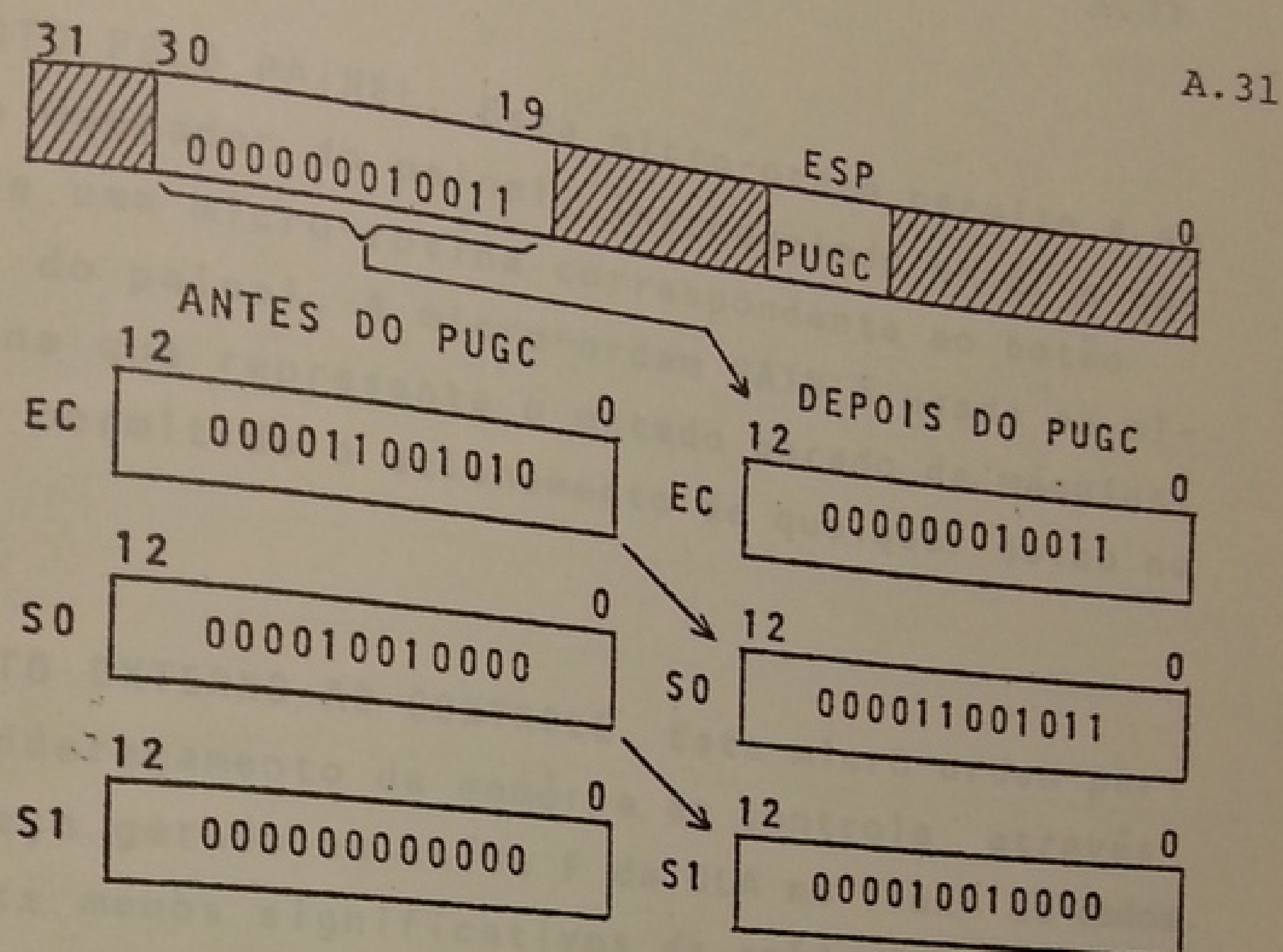


PUGC PULO PARA MICRO-SUBROTINA. Esta micro-instrução é utilizada para a execução de pulos para micro-subrotina nos micro-programas, pois além de passar a execução para a posição especificada por DC <19 : 30> a micro-ordem PUGC guarda o endereço de retorno da micro-instrução seguinte, numa pilha de dois registradores S0, S1. Como só existem dois registradores na pilha, pode-se ter no máximo dois níveis encadeados de micro-subrotinas nos micro-programas de controle.

Esta micro-ordem armazena o conteúdo de EC incrementado de 1 no registrador de pilha S0, tendo antes armazenado S0 em S1, perdendo o conteúdo anterior de S1. Logo após, é feita a transferência dos bits <19 : 30> do registrador DC para o registrador EC, e é incrementado o contador de subrotinas.

Essa micro-instrução está condicionada ao teste determinado no campo COND.

Exemplo:



As micro-ordens PUGC e FIM iniciam e finalizam, respectivamente, a chamada de uma micro-subrotina.

FOR1 FORÇA MAPEADOR DE ENDEREÇAMENTO MO. Esta micro-ordem é usada ao término da micro-rotina de busca, permitindo a seleção do mapeador de endereçamento MO, e introduzindo no registrador EC o endereço selecionado para a micro-rotina de cálculo de endereço efetivo de MO, ou eventualmente uma rotina de preparação de dados. Os registradores S0 e S1 e o contador de subrotinas são feitos iguais a zero.

FOR2 FORÇA MAPEADOR DE ENDEREÇAMENTO ME OU DE EXECUÇÃO. Esta micro-ordem é usada ao término das micro-rotinas de endereçamento MO para selecionar o mapeador de endereçamento ME ou de execução da instrução. Se a instrução for curta, o registrador EC é carregado com um endereço de uma micro-rotina de execução. Se for longa, EC é carregado inicialmente com uma micro-rotina de cálculo de endereço efetivo ME, no final da qual, uma micro-ordem FOR2 seleciona o endereço de uma micro-rotina de execução.

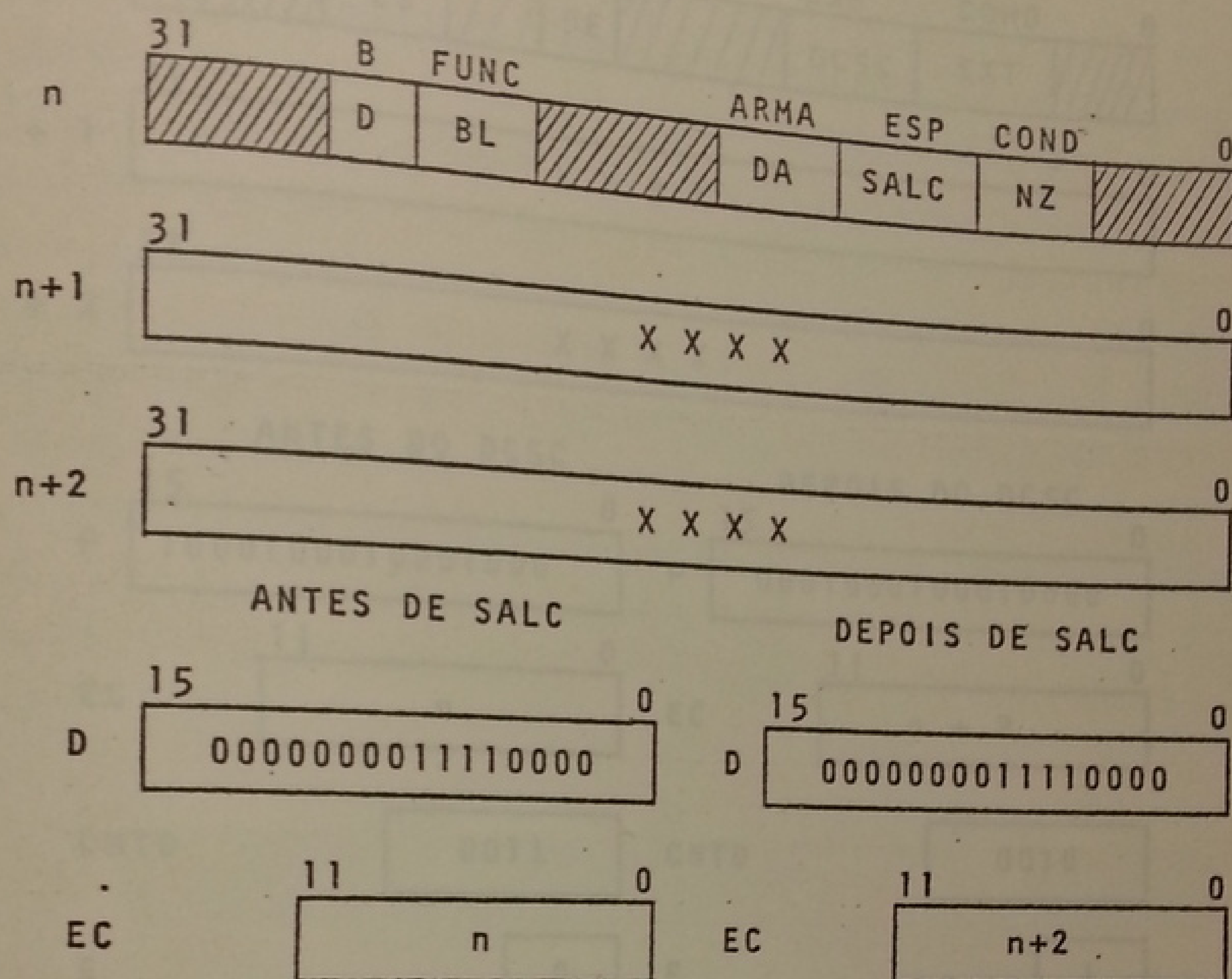
PAIN MAPEAMENTO PELO PAINEL. Esta micro-ordem permite a seleção do mapeador do painel, introduzindo em EC o endereço de uma micro-rotina correspondente ao botão acionado do painel. A micro-ordem PAIN é usada na micro-rotina que representa o estado parado da máquina, quando é permitido o acionamento de qualquer botão no painel.

FUNC MAPEAMENTO EXTERNO AO CONTROLE. Esta micro-ordem permite o endereçamento da memória de controle através do endereço gerado na saída F da ULA no fluxo de dados. Os 12 bits menos significativos da saída F da ULA são carregados em EC, e um micro-programa é normalmente endereçado e executado.

A micro-ordem FUNC é particularmente útil quando da introdução de uma micro-rotina correspondente a uma nova instrução ou micro-instrução por parte do usuário na memória de controle. Nesse caso, o usuário deve conhecer a posição correta dessa micro-rotina, já que o controle de endereçamento da memória "ler-somente " está em seu poder.

SALC SALTO CONDICIONAL. Esta micro-ordem provoca o salto da micro-instrução seguinte àquela que contém a micro-ordem SALC, caso a condição especificada no campo COND seja satisfeita.

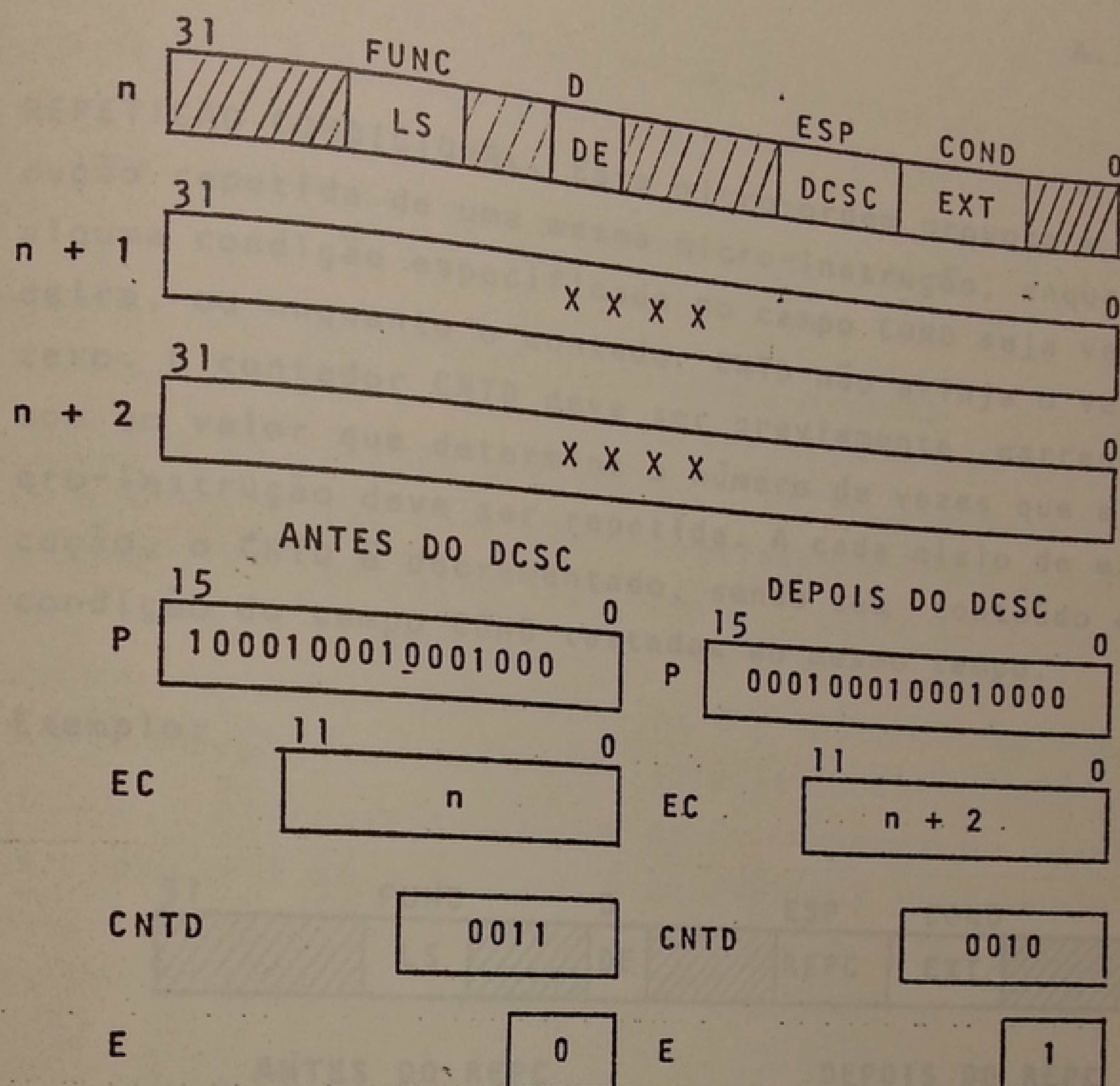
Exemplo:



No exemplo acima, na micro-instrução n, é feito um teste para verificar se o conteúdo do registrador D é zero. Como D contém um valor não nulo, a próxima micro-instrução (n+1) não é executada.

DCSC DECREMENTO E SALTO SE CONDIÇÃO VERDADEIRA. Esta micro-ordem provoca o decremento de 1 do contador CNTD. Caso a condição especificada no campo COND seja satisfeita, a próxima micro-instrução será saltada.
Se não for declarada nenhuma condição no campo COND (isto é, NOP), o contador CNTD será decrementado, e a próxima instrução será saltada somente se o contador CNTD for nulo.
Verifica-se, portanto, que o saldo ocasionado será devido a um "ou" entre a condição especificada em COND e a condição ZCNT (zero no contador CNTD).

Exemplo :

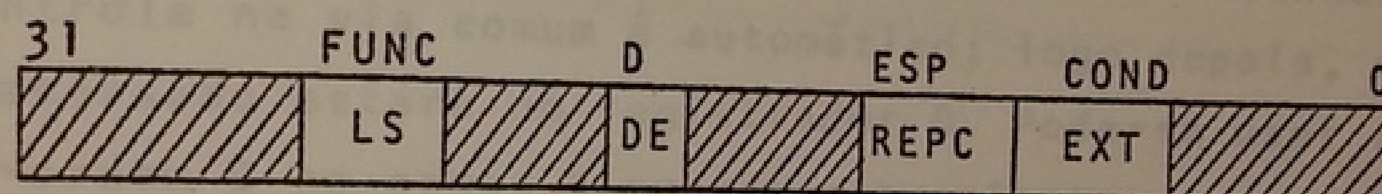


Nesse exemplo, o registrador deslocador P é deslocado de uma posição à esquerda, atualizando o bit E de extensão. O contador CNTD é decrementado. A seguir, tanto o contador como a extensão são testados. A próxima micro-instrução é saltada pois E contém o valor 1.

SEVI SELEÇÃO DO VI. Esta micro-ordem auxiliar é usada nos micro-programas de instruções de entrada e saída antes de se efetuar uma leitura do vetor de interrupção de um canal. O sinal de controle gerado por essa micro-ordem age na via comum permitindo, num diálogo entre o processador e o canal, a transferência de informações sobre o estado do canal e do periférico que pediram interrupção.

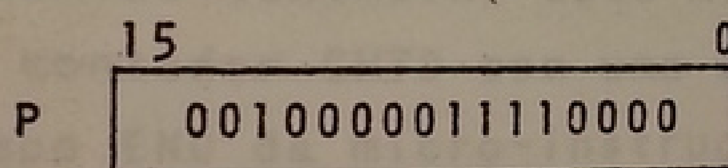
REPC REPETIÇÃO CONDICIONAL. Esta micro-ordem provoca a execução repetida de uma mesma micro-instrução, enquanto alguma condição especificada no campo COND seja verdadeira, ou enquanto o contador CNTD não atinja o valor zero. O contador CNTD deve ser previamente carregado com um valor que determine o número de vezes que a micro-instrução deve ser repetida. A cada ciclo de execução, o CNTD é decrementado, sendo seu conteúdo e a condição do campo COND testadas ao mesmo tempo.

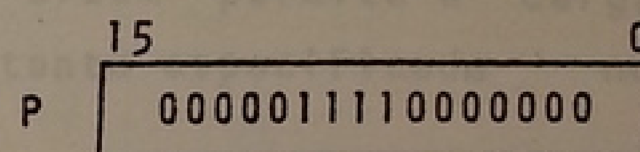
Exemplo:



ANTES DO REPC

DEPOIS DO REPC

P


P


CNTD 0100

CNTD 0001

E 0

E 1

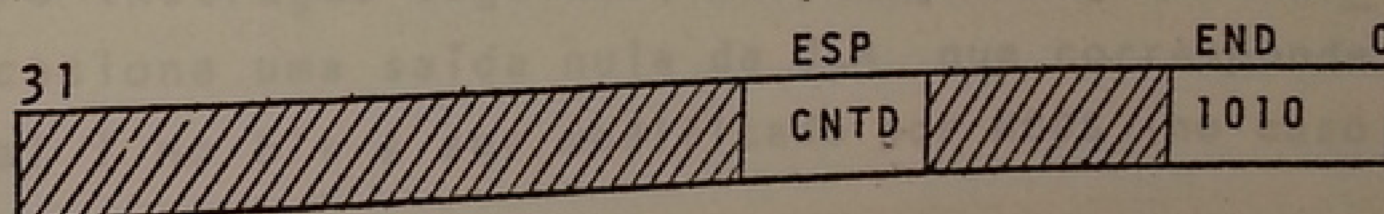
O registrador P é deslocado para a esquerda até que o contador assuma o valor zero, ou até que a extensão E esteja com o valor 1.

LE LEITURA DA MEMÓRIA. Esta micro-ordem inicia um ciclo de acesso à memória principal, para a leitura de um dado cujo endereço está especificado no registrador E. O dado lido é armazenado no registrador D.

RVC REQUISIÇÃO DA VIA COMUM. Esta micro-ordem determina uma requisição de uso da via comum quando do acesso à unidade de entrada e saída do G-10. Os registradores I e E devem ser carregados previamente com a primeira palavra da instrução (por exemplo: ENTRA DADO, SAI DADO, etc), e o registrador D com o dado (no caso SAI DADO), ou máscara (no caso de uma instrução de função ou comando de entrada e saída). Feito isso, deve ser usada a micro-ordem RVC; o posicionamento das linhas de controle na via comum é automático; logo depois, o dado acessado estará no registrador D. Pode-se usar a micro-ordem D no campo R quando conveniente.

CNTD CARGA DO CONTADOR. Esta micro-ordem permite a carga do contador CNTD com uma constante especificada no campo END da micro-instrução.

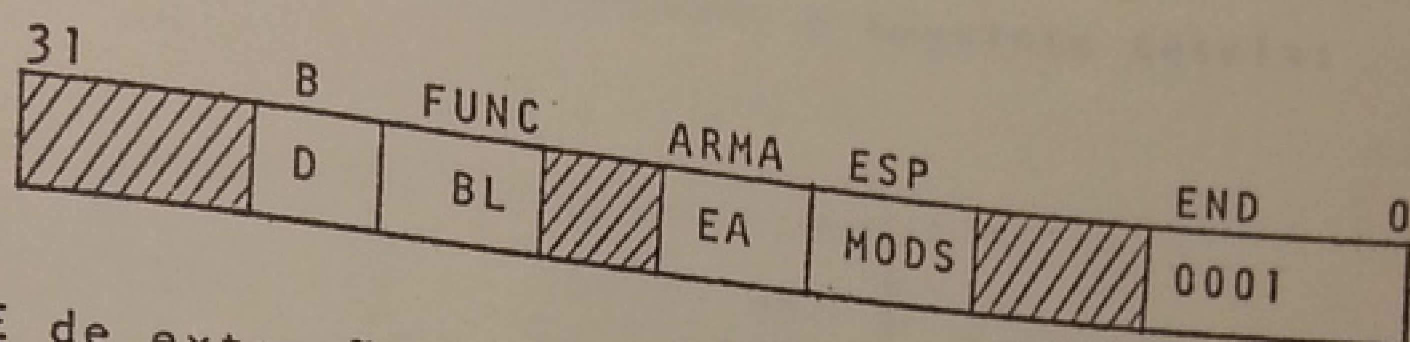
Exemplo:



O contador CNTD é carregado com a constante 1010.

MODS MODIFICAÇÃO DO REGISTRADOR DE ESTADO. Esta micro-ordem permite a atualização de um dos bits do byte direito do registrador de estado S. Esse bit vem especificado no campo END.
No caso dos bits E, T, S, Z e P, cada um deles pode ser atualizado individualmente usando a micro-ordem MODS no campo ESP e endereçando o bit no campo END da mesma micro-instrução que opera aritmética ou logicamente a ULA.

Exemplo:



O bit E de extensão é atualizado segundo a operação lógica realizada. O resultado final é E = 0.

No caso dos bits M, I e C, estes são individualmente feitos iguais a 1 ou 0, dependendo de uma operação na ULA, cujo resultado é nulo ou não.

Exemplo:

1	/0000 P	O registrador P é carregado com /0000
2	P → P,MODS$(0001)_2$	O bit 1 (bit E)é feito igual a 1

Na primeira micro-instrução o registrador P é carregado com zero.

Na micro-instrução seguinte, a operação lógica realizada ocasiona uma saída nula da ULA que corresponde ao nível 1 no bit a ser modificado por MODS (no caso bit E).

Se o registrador P contivesse um dado não nulo,a saída da ULA, na segunda micro-instrução,corresponderia ao nível 0 no bit a ser modificado por MODS.

<u>Observação</u>:Não se deve usar MODS numa micro-instrução cuja função lógica ou aritmética atualize os bits do registrador de estado E, T, S, Z e P, pois estes são alterados simultânea e automaticamente com este tipo de operação.

O endereçamento aos bits do registrador de estado deve constar no campo NED, segundo a seguinte tabela:

BIT DO REGISTRADOR DE ESTADO	CÓDIGO NO CAMPO END
T	$(0000)_2$
E	$(0001)_2$
S	$(0010)_2$
Z	$(0011)_2$
P	$(0100)_2$
M	$(0101)_2$
I	$(0110)_2$
C	$(0111)_2$

ESC ESCRITA NA MEMÓRIA. Esta micro-ordem inicial um ciclo de escrita na memória principal. O dado a ser escrito deve ser previamente carregado no registrador D e seu endereço, no registrador E. Pode-se usar a micro-ordem D no campo R se tal for conveniente.

RECI RECONHECE INTERRUPÇÃO. Esta micro-ordem auxiliar é usada para gerar um sinal de reconhecimento enviado pelo processador central ao canal que pediu a interrupção, durante a execução da micro-rotina de tratamento associado a esse canal.

LINT LIMPA INTERRUPÇÃO. Esta micro-ordem auxiliar é usada na geração de um sinal para limpar a interrupção externa correspondente ao nível que acabou de ser tratado durante um retorno de interrupção.

PERI PERMITE INTERRUPÇÃO. Esta micro-ordem auxiliar é usada na geração de um sinal para liberar o sistema de interrupção após o reconhecimento da última interrupção.

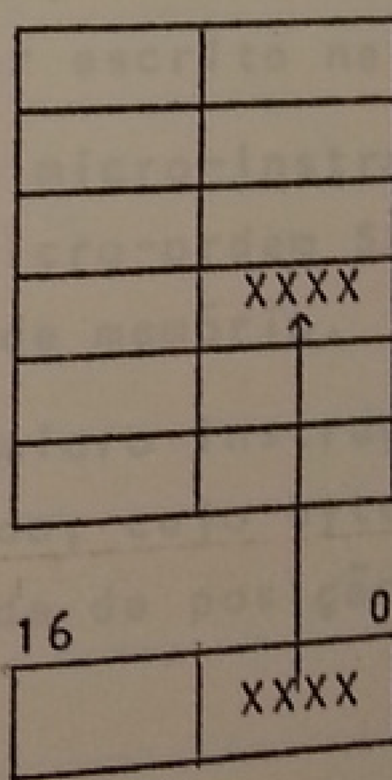
EEX MICRO-ORDEM ESPECIAL DE MAPEAMENTO. Esta micro-ordem auxiliar permite a mudança para o endereço da microrotina de execução das instruções longas após o cálculo do endereço efetivo ME. Nesse caso EEX deve ser usada antes da micro-ordem FOR2.

SETE SELEÇÃO DO BYTE ESQUERDO DA MEMÓRIA. Esta micro-ordem auxiliar é usada para permitir a operação de armazenamento no byte esquerdo de uma certa posição na memória principal, sem alterar o byte direito. O endereço desta posição deve estar no registrador E que foi previamente carregado, e o dado deve estar no registrador D.

Normalmente, nas instruções de bytes, quando se deseja armazenar um byte na memória, deve se prosseguir da seguinte maneira :

1) Armazenamento no byte direito da memória.

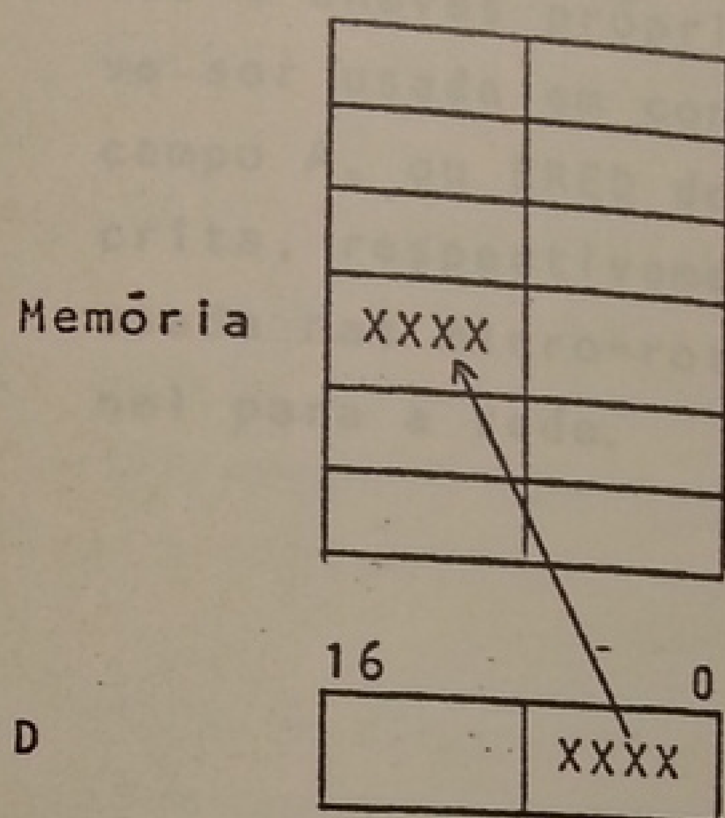
Memória



D

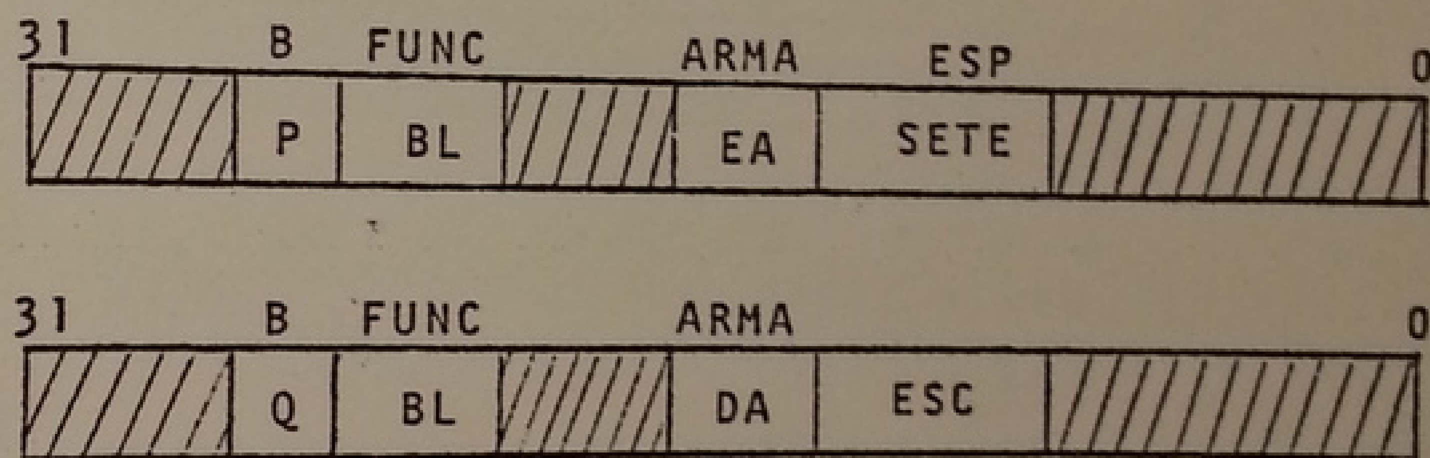
O byte direito do registrador D é carregado no byte direito da posição de memória, automaticamente, com a micro-ordem ESC. O byte esquerdo não é alterado.

2) Armazenamento no byte esquerdo da memória.



O byte direito do registrador D é carregado no byte esquerdo da posição de memória, com a micro-ordem SETE seguida da micro-ordem ESC. O byte direito não é alterado.

Exemplo :



Supõe-se que, inicialmente, o endereço esteja em P e o dado a ser escrito na memória em Q.

Na primeira micro-instrução, o registrador E é carregado, e a micro-ordem SETE seleciona o byte esquerdo da posição de memória.

Na segunda micro-instrução, o registrador D é carregado com o dado, cujo byte direito será armazenado no byte esquerda da posição da memória, com a micro-ordem ESC.

PAR ENDEREÇAMENTO PAINEL-REDE. Esta micro-ordem auxiliar permite o endereçamento dos registradores da rede pelas 4 chaves próprias do painel. A micro-ordem PAR deve ser usada em conjunto com as micro-ordens SRED do campo A, ou ERED do campo ARMA, para uma leitura ou escrita, respectivamente, na rede. Normalmente, PAR é usada nas micro-rotinas que tratam dos botões do painel para a rede.

A.9 CAMPO COND

Este campo contêm as condições que devem ser testadas com as micro-ordens condicionais do campo ESP.

O campo COND é constituído pelos bits <4 : 6> da palavra de controle, mais o bit <31> correspondente ao campo R. Dessa forma, ao ser usada uma micro-ordem condicional em ESP, o sentido do campo R não é levado em conta, passando este a constituir o quarto bit do campo COND.

Uma condição é dita verdadeira quando qualquer uma das condições abaixo assume o valor 1.



(0	$000)_2$ - NOP
(0	$001)_2$ - EXT
(0	$010)_2$ - NT
(0	$011)_2$ - NZ
(0	$100)_2$ - POS
(0	$101)_2$ - US
(0	$011)_2$ - NQI
(0	$111)_2$ - PAR
(1	$000)_2$ - LONG
(1	$001)_2$ - IND
(1	$010)_2$ - INX
(1	$011)_2$ - NINT
(1	$100)_2$ - NCHA
(1	$101)_2$ - ZCNT
(1	$110)_2$ - BP
(1	

NOP — NENHUMA OPERAÇÃO. Nenhuma condição é testada, portanto a micro-ordem do campo ESP passa a ser executada incondicionalmente, exceção feita às micro-ordens REPC e DCSC do campo ESP que serão condicionadas a zero no contador CNTD, implicitamente.

EXT — EXTENSÃO. EXT = 1 significa que houve um "vai-um" resultante da operação efetuada na ULA ou no deslocador.

Esta condição só pode ser testada na mesma micro-instrução da operação, pois ela é gerada e é válida durante um só ciclo de controle. EXT não representa necessariamente o valor do bit E do registrador S, e portanto seu teste posterior é inválido.

NT — TRANSBORDAMENTO. NT = 1 significa que não houve transbordamento na operação efetuada na ULA ou no deslocador, e portanto a micro-ordem condicional do campo ESP será executada.

Esta condição só pode ser testada na mesma micro-instrução da operação, pois ela é gerada e é válida durante um só ciclo de controle. NT não indica necessariamente qual seja o valor do bit T do registrador S, e portanto seu teste posterior é inválido.

NZ — ZERO. NZ = 1 significa que o resultado da operação efetuada na ULA é diferente de zero, e portanto a micro-ordem condicional do campo ESP será executada.

Esta condição só pode ser testada na mesma micro-instrução da operação, pois ela é gerada e é válida durante um só ciclo de controle. NZ não indica necessariamente qual seja o valor do bit Z do registrador S, e portanto seu teste posterior é inválido.

POS POSITIVO. POS = 1 significa que o resultado da operação efetuada na ULA é positivo ou nulo, sendo o bit < 15 > do mesmo igual a 0.

Esta condição só pode ser testada na mesma micro-instrução da operação, pois ela é gerada e é válida durante um só ciclo de controle. POS não indica necessáriamente qual seja o valor do bit S do registrador de estado,e portanto seu teste posterior é inválido.

US MODO USUÁRIO / SUPERVISOR. US = 1 significa que o modo de processamento corrente é o de Usuário. Se US =0, o modo é de Supervisor. Esta informação está diretamente ligada ao bit M do registrador S e pode ser testada em qualquer instante.

NQ1 BIT <1 > DE Q. NQ1 = 1 significa que o bit Q <1> do registrador Q assume o valor 0, e portanto a micro-ordem condicional do campo ESP será executada. NQ1 pode ser testada a qualquer instante conveniente no micro-programa.

PAR RESULTADO PAR / IMPAR. PAR = 1 significa que o resultado da operação na ULA é ímpar, sendo o bit < 0 > do mesmo igual a 1. Portanto a micro-ordem condicional no campo ESP será executada.

Esta condição só pode ser testada na mesma micro-instrução da operação, pois ela é gerada e é válida durante um só ciclo de controle . PAR não indica necessariamente qual seja o valor do bit P do registrador S, e portanto seu teste posterior é inválido.

LONG INSTRUÇÃO LONGA. LONG = 1 significa que a instrução que está sendo executada é longa e portanto a micro-ordem do campo ESP será executada.

IND INDIREÇÃO. IND = 1 significa que o campo MO da instrução de máquina corrente assume valores indiretos$(01)_2$, $(10)_2$ ou $(11)_2$. A micro-ordem condicional no campo ESP será executada.

INX INDEXAÇÃO. INX = 1 significa que o cálculo de endereço efetivo ME da instrução de máquina corrente inclui a indexação, e portanto a micro-ordem condicional no campo ESP será executada.

NINT INTERRUPÇÃO. NINT = 1 significa que não há pedido de interrupção pendente, e portanto a micro-ordem condicional no campo ESP será executada.

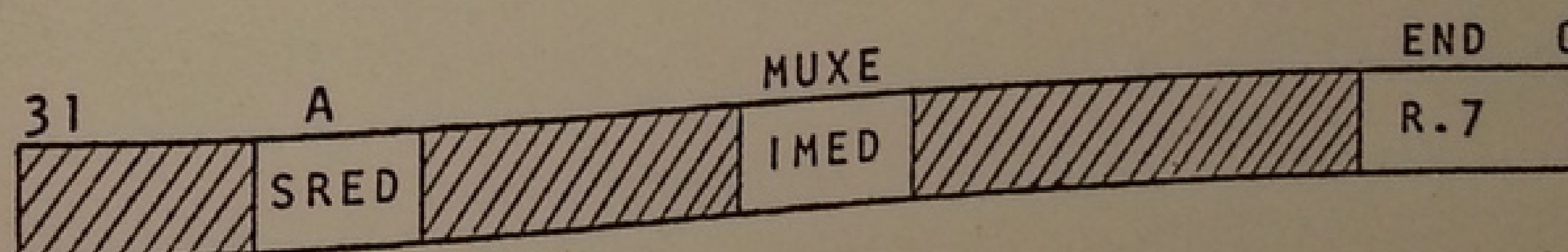
NCHA CHAVES. NCHA = 1 significa que nenhuma chave ou botão de controle do painel foi acionada, e portanto a micro-ordem condicional do campo ESP será executada.

ZCNT ZERO NO CONTADOR. ZCNT = 1 significa que o contador CNTD contém zero, e portanto a micro-ordem condicional no campo ESP será executada.

BP BASE DE PROGRAMA. BP = 1 significa que a instrução de máquina em execução se refere à base de programa, e portanto a micro-ordem condicional no campo ESP será executada.

MNEMÔNICOS NO CAMPO END	CÓDIGO	REGISTRADOR
R.0	$(0000)_2$	SP0 - Registrador de Trabalho 0
R.1	$(0001)_2$	SP1 - Registrador de Trabalho 1
R.2	$(0010)_2$	SP2 - Registrador de Trabalho 2
R.3	$(0011)_2$	SP3 - Registrador de Trabalho 3
R.4	$(0100)_2$	LD - Limite de Dados
R.5	$(0101)_2$	LP - Limite de Programa
R.6	$(0110)_2$	BD - Base de Dados
R.7	$(0111)_2$	BP - Base de Programa
R.8	$(1000)_2$	R0 - Contador de instruções CI
R.9	$(1001)_2$	R1 - Ponteiro da pilha
R.A	$(1010)_2$	R2 } Registradores gerais
R.B	$(1011)_2$	R3 } Registradores gerais
R.C	$(1100)_2$	R4 } Registradores gerais
R.D	$(1101)_2$	R5 } Registradores gerais
R.E	$(1110)_2$	R6 } Registradores gerais
R.F	$(1111)_2$	R7 } Registradores gerais

Exemplo:



O registrador de Base de Programa é selecionado para ser operado.

A.10 CAMPO END

Este campo tem basicamente a função de selecionar um dos registradores da rede, vetor de interrupção, registrador de chaves do painel e registrador E.

Uma outra função alternativa é de gerar uma constante a ser carregada no contador CNTD com a micro-ordem CNTD do campo ESP, ou de conter os 4 bits menos significativos de uma constante a ser carregada no registrador P com a micro-ordem CAR do campo D; nesse caso, esta constante é gerada pelos bits <19 : 30> e <0 : 3> da palavra de controle.

Os elementos a que a micro-ordem se relaciona são os seguintes:

1) Registradores da rede

Esses registradores são endereçados no campo END e selecionados com as micro-ordens SRED do campo A, IMED do campo MUXE, e ERED do campo ARMA.

2) Registrador de Endereços, Chaves do Painel e Vetor de Interrupção.

Esses registradores são endereçados pelo campo END e selecionados pela micro-ordem C do campo A.

MNEMÔNICOS NO CAMPO END	CÓDIGO	REGISTRADOR
EE	$(0001)_2$	Registrador de Endereços E
PNL	$(0010)_2$	Chaves do PAINEL
VI	$(1000)_2$	Vetor de Interrupção

Exemplo:

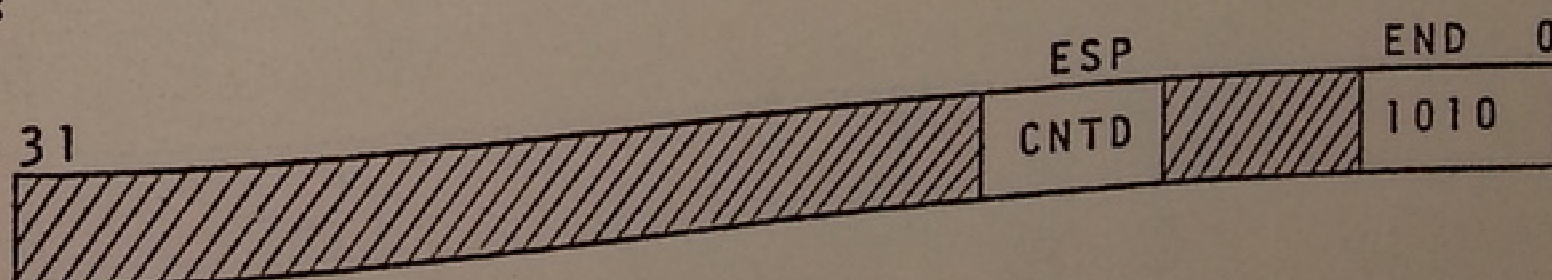


O Registrador de Chaves do Painel é selecionado para ser operado.

3) Carga de uma constante no contador CNTD

O contador CNTD de 4 bits é carregado com a constante do campo END, com a micro-ordem CNTD do campo ESP.

Exemplo:



CNTD é carregado com a constante $(1010)_2$.

APÊNDICE B - EXEMPLOS DE MICROPROGRAMAS DO G-10

```
**BLT   BBLLTT
**
**
** IA MEMORIA
**
**
BLSCA        SREC
             SREC   G     ICR
             SREC   G     SOM    IMED  CAR              R.8
                          SUB    IMED        GA   SEBL  R.7
                                 IMED        EA   LE    R.5
      . D           G     BL                      SALC  EXT        PRCTM
             SREC                IMED             PLAC
                          ICR    IMED        ERED             R.8
                                       CAR   GA   SALC  LONG       R.8
 **                                    CAR        FCR1             COFF
             SREC   G     SOM    IMED
             SREC   G     SUB    IMED        EA   LE               R.7
                                                  SALC  EXT        R.5
        D           G     BL     IMED             PLAC             PRCTM
 **                                          ERED FCR1             R.8
 **
 ** ROTINAS DE ENDERECAMENTC DE REGISTRADCR
 **
 **
 ** ENDERECC DIRETC E INDIRETC
 **
 **
MCDI         SREC         A1     DEST  CAR   PA   SALC  INC
                    P     BL     IMED        ERED FCR2             R.2
                    P     BL     IMED        ERED FCR2             R.1
 **
 **
 ** ENDERECC INDIRETO PRE-INCREMENTACC
 **
 **
MINC         SREC         ICR    DEST  CAR   PA
                    P     BL     IMED        ERED                  R.1
                    P     BL     DEST        ERED FCR2
 **
 **
 ** ENDERECC INDIRETO POS-DECREMENTACC
 **
 **
MDEC         SREC         A1     DEST  CAR   PA
                    P     BL     IMED        ERED                  R.1
             SREC         DCR    DEST        ERED FCR2
 **
 ** CALCLLO DE ENDERECC EFETIVO DE DESLOCAMENTC
 **
 **
 ** DIRETC
 **
 **
MEDIR        SREC         A1     IMED  CAR   PA   SALC  BP         R.7
             SREC         A1     IMED  CAR   PA                    R.6
                    F     BL     IMED        ERED                  R.0
 **
MEDI1                                             PLGC  INX        INDIX
 **
ALX          SREC   D     SOM    IMED  CAR   PA   EEX              R.0
                    P     BL                 EA   FCR2
 **
```

```
**
** INDIRETO UM NIVEL PRE-INDEXADO
**
**
MEIP1
                                                          PLGC  INX        INDIX
                                                          PLGC             CATAB
**                                                        PLAC             AUX
**
** INDIRETO DOIS NIVEIS PRE-INDEXADO
**
**
MEIP2
                                                          PLGC  INX        INDIX
                                                          PLGC             CATAB
                                                          PLGC             CATA
**                                                        PLAC             AUX
**
** INDIRETO UM NIVEL POS-INDEXADO
**
**
MEI1P
                                                          PLGC             CATAB
                                                          PLAC             MEC11
**
**
** ROTINA AUXILIAR IMEDIATAS CURTAS
**
**
IMEC          I       P     EL                 DA   FCN2
**
**
** ROTINA AUXILIAR DESLOCAMENTO CURTO
**
**
LGAC                  DP    BL                 CA
              SREC    D     SOM   IMED  CAR    GA                  K.A
              SREC    G     SOM   IMED         EA   LE             K.6
                                                    PLGC           PROTO
**
**
** ROTINA AUXILIAR PARA INSTRUCOES COM ME E SEM MC
**
**
SMC                         BL                      FCN2
**
**
** EXECUCAO INSTRUCOES IMEDIATAS CURTAS
**
**
** EXECUCAO INSTRUCOES COM DESLOCAMENTO CURTO
**
**
** EXECUCAO CARGA IMEDIATA CURTA
**
**
** EXECUCAO CARGA DESLOCAMENTO CURTO
**
**
CARC                  D     BL    IMED         EREO                K.F
              SREC          A2S   IMED         EREO FIM            K.F
**
```

```
**
** EXECUCAO SOMA IMEDIATA CURTA
**
**
** EXECUCAO SOMA DESLOCAMENTO CURTO
**
**
SOMO            SRED  D      SOMS  IMED              ERED  FIM                         R.F
**
**
** EXECUCAO SUBTRACAO IMEDIATA CURTA
**
**
** EXECUCAO SUBTRACAO DESLOCAMENTO CURTO
**
**
** EXECUCAO OU LOGICO IMEDIATA CURTA
**
**
** EXECUCAO OU LOGICO DESLOCAMENTO CURTO
**
**
** EXECUCAO E LOGICO IMEDIATA CURTA
**
**
** EXECUCAO E LOGICO DESLOCAMENTO CURTO
**
**
** EXECUCAO OU EXCLUSIVO IMEDIATA CURTA
**
**
** EXECUCAO OU EXCLUSIVO DESLOCAMENTO CURTO
**
**
TIPO            SRED  D      TIPS  IMED              ERED  FIM                         R.F
**
**
** EXECUCAO COMPARA IMEDIATA CURTA
**
**
** EXECUCAO COMPARACAO DESLOCAMENTO CURTO
**
**
COMPO           SRED  D      SUBS  IMED                    FIM                         R.F
**
**
** EXECUCAO INSTRUCOES COM REFERENCIA A MEMORIA
**
**
** EXECUCAO CARGA (M)--(R)
**
**
CAREM                                                      FLGC                        LEM
                      D      SOMS          CAR   GA        SALC  INC
                      G      BL     DEST         ERED      FIM
                                                           FLAC                        ESCR
**
**
** EXECUCAO ARMAZENA (R)--(M)
**
**
```

```
ARMM

**                                                   PUGC              FRCTX
GUARM                                                PUGC   INC        FREFC
           SRED   P       BL
       D                  A1    IMED           EA
**                        BL                   DA    ESC               R.2
**                                                   FIM
** EXECUCAO COMPARA (R)=(M)
**
**
COMPM

                D         BL              CAR   PA   PUGC              LEM
           SRED P         SUBS  IMED                 PUGC   INC        FREFC
**                                                   FIM               R.2
**
** EXECUCAO SOMA (R)+(M)=(R)
**
**
SOMM

                D         BL              CAR   PA   PUGC              LEM
           SRED P         SOMS  DEST            ERED SALC   INC
                                                     FIM
           SRED P         SOMS  IMED                 PUGC              FREFC
       D                                        DA   ESC               R.2
**                        BL                         FIM
**
** EXECUCAO SUBTRACAO (R)-(M)=(R)
**
**
SUBM

                D         BL              CAR   PA   PUGC              LEM
           SRED P         SUBS  DEST            ERED SALC   INC
                                                     FIM
           SRED P         SUBS  IMED                 PUGC              FREFC
       D                                        DA   ESC               R.2
**                        BL                         FIM
**
** EXECUCAO OU LOGICO  (R)OU(M)--(R)
**
**
OUM
                D         BL              CAR   PA   PUGC              LEM
           SRED P         OULS  DEST            ERED SALC   INC
                                                     FIM
           SRED P         OULS  IMED                 PUGC              FREFC
       D                                        DA   ESC               R.2
**                        BL                         FIM
**
** EXECUCAO E LOGICO (M) E (R)=(R)
**
**
EM
                D         BL              CAR   PA   PUGC              LEM
           SRED P         ELS   DEST            ERED SALC   INC
                                                     FIM
           SRED P         ELS   IMED                 PUGC              FREFC
       D                                        DA   ESC               R.2
**                        BL                         FIM
**
```

```
**
** CARREGADOR MICRO PROGRAMADO
**
ALPAL                                           CAR                              0000
                     P       BL      IMED             ERED                       R,6
                                                            PUGC                 ENCHE1
**
**
** LEITURA DO NUMERO DE PALAVRAS
**
**
ALP1                 D       BL                       CA    SALC   NZ
                                                            PLAC                 XX
                                                            PUGC                 BYTE1
               SRED  D       SOM     IMED             ERED                       R,6
                     D       BL      IMED             ERED                       R,7
**
**
** LEITURA DA POSICAO INICIAL NA MEMORIA
**
INMEM                                                       PUGC                 GERAL
               SRED  D       SOM     IMED             ERED                       R,6
                     D       BL              CAR      GA
**
** LEITURA DO CARREGADOR ABSOLUTO
**
LOOPC                                                       PUGC                 GERAL
               SRED  D       SOM     IMED             ERED                       R,6
                     G       BL                       EA    ESC
        D            G       ADI             CAR      GA
               SRED          ICR     IMED             ERED  SALC   NZ            R,7
                             BL                             SALC
                                                            PLAC                 LOOPC
R0                                                          PUGC                 GERAL
               SRED  D       SOM     IMED             ERED                       R,6
                     D       BL      IMED             ERED                       R,8
               SRED          DCR     IMED             ERED                       R,8
R1                                                          PUGC                 GERAL
               SRED  C       SOM     IMED             ERED                       R,6
                     C       BL      IMED             ERED                       R,9
**
** TESTE DO CHECKSUM
**
OUT                                                         PUGC                 GERAL
                                             CAR                                 FFFF
               SRED  D       SUB     IMED             ERED  SALC   NZ            R,6
                                                            PLAC                 CERTO
                     P       BL                       EA
                     P       BL                       DA
CERTO                P       BL      IMED             ERED                       R,5
                     P       BL      IMED             ERED                       R,4
                             A1              CAR      PA    MODS                 0005
                     P       BL      IMED             ERED                       R,7
                     P       BL      IMED             ERED                       R,6
                                                            PLAC                 CHAV
**
** DEFINICAO DAS INSTRUCOES DE ENTRADA E SAIDA
**
ENCHE                                        CAR                                 28EC
                     P       BL      IMED             ERED                       R,8
                                             CAR                                 0004
```

```
                          P     BL    IMED              ERED                       R.C
                          P     BL    IMED       CAR                               3000
**                                                      ERED                       R.0
** INICIALIZACAO DO DISPOSITIVO
**
INIC
                          P     BL               CAR                               2009
                          P     BL                      EA
                                                        IA
          D               P     BL               CAR                               0024
**                                                      DA     RVC
** LEITURA DE UM BYTE
**
LOOPD            SRED           A1    IMED
          D      SRED           A1    IMED              IA                         R.B
                 SRED     D     EL    IMED              EA     RVC                 R.B
LOOPD1           SRED           A1    IMED                     SALC   NZ           R.C
                                                        EA     SALC                R.D
                 SRED           A1    IMED                     PLAC                LOOPD
          D                     BL                      IA                         R.D
                                                               RVC
                                                 CAR                               00FF
                          P     BL    IMED              ERED                       R.F
                 SRED     D     EL    IMED              DA     FIM                 R.F
**
** LEITURA DE DOIS BYTES E COMPACTACAO
**
GERAL                                                          PUGC                INICE
**
** LEITURA DO BYTE MAIS SIGNIFICATIVO
**
BYTE1                     D     BL               CAR    PA     CNTD                0008
                                LS               DE            REFC   ZCNT
                          P     BL    IMED              ERED                       R.0
**
                                                               PUGC                INICE
                 SRED     D     SOM   IMED              DA     FIM                 R.0
**
XX                                                             PUGC                INICE
                                                               PLAC                NUF1
**
**
**
**
```

APÊNDICE C - EXEMPLOS DE MICROPROGRAMAS DO EMULADOR DO IBM 1130

```
**BLT   BBLLTT
**
** BUSCA
**
BUSCA                         BL                            SEEL
          D    SRED           A1    IMED           EA       LE                   R,8
               SRED           ICR   IMED           ERED     SALC    LONG         R,8
               SRED    DP     SCM   CRGM           EA       FCR2
               SRED           A1    IMED           EA       LE                   R,8
          D                   BL                            FCR2
**
** LONGA DIRETA NAO INDEXADA
**
L1                     D      BL                   EA       EEX
               SRED           ICR   IMED           ERED     FCR2                 R,8
**
** LONGA DIRETA INDEXADA
**
L2             SRED    D      SCM   CRGM           EA       EEX
               SRED           ICR   IMED           ERED     FCR2                 R,8
**
** LONGA INDIRETA NAO INDEXADA
**
L3                     D      BL                   EA       LE
                              BL                            EEX
          D    SRED           ICR   IMED           ERED     FCR2                 R,8
**
** LONGA INDIRETA INDEXADA
**
L4             SRED    D      SCM   CRGM           EA       LE
**
** CURTA
**
CURTA                         BL                            EEX
          D    SRED           ICR   IMED           ERED     FCR2                 R,8
**
** LOAD ACC
**
LD        D            D      BL                   EA       LE
                       D      BL    IMED           ERED     FIM                  R,E
**
** LOAD DOUBLE
**
LDD                                                         PUGC                 LD
          D    C              ICR                  EA       LE             EE
                       D      BL    IMED           ERED     FIM                  R,F
**
** STORE ACC
**
STC                    D      BL                   EA
               SRED           A1    IMED           DA       ESC                  R,F
          D                   BL                            FIM
**
** STORE DOUBLE
**
STD                                                         PLGC                 STD
               C              ICR                  EA                      EE
               SRED           A1    IMED           DA       ESC                  R,E
          D                   BL                            FIM
**
** LOAD INDEX
```

```
**
LDX       D           D    BL                 EA    LE
                      DP   BL     CRGM          ERED  SALC  LONG
                           BL                         FIM
                      D    BL     CRGM          ERED  FIM
**
** STORE INDEX
**
STX                   D    BL                 EA
                SRED       A1     CRGM          DA    ESC
          D                BL                         FIM
**
** STORE STATUS
**
STS                   D    BL                 EA    LE
                                         CAR                          0003
                      P    BL     IMED          ERED                  R.1
                SRED  S    EL     IMED          ERED                  R.1
                                         CAR                          FF00
          D           P    BL     IMED          ERED                  R.2
                SRED  D    EL     IMED          DA                    R.2
                SRED  D    DULS   IMED          DA    ESC             R.1
          D                BL                         FIM
**
** LOAD STATUS
**
LDS                   DP   DUL                  SA    FIM
**
** ADD
**
A         D           D    BL                 EA    LE
                SRED  D    SOMS   IMED          ERED  FIM             R.E
**
** ADD DOUBLE
**
AD        D           D    BL            CAR  EA    LE
                      D    BL            CAR  FA
                C          ICR                  EA                EE
          D                BL                         LE
                SRED  D    SOMS   IMED          ERED  SALC  EXT       R.F
                SRED  P    SOMS   IMED          ERED  FIM             R.E
                SRED  P    ADS    IMED          ERED  FIM             R.E
**
** SUBTRACT
**
S         D           D    BL                 EA    LE
                SRED  D    SUBS   IMED          ERED  FIM             R.E
**
** SUBTRACT DOUBLE
**
SD        D           D    BL                 EA    LE
                      D    BL            CAR  FA
                C          ICR                  EA                EE
          D                BL                         LE
                SRED  D    SUBS   IMED          ERED  SALC  EXT       R.F
                SRED  P    SUBS   IMED          ERED  FIM             R.E
                SRED  P    DIMS   IMED          ERED  FIM             R.E
**
** AND
**
AND       D           D    BL                 EA    LE
```

```
                 SREC  D        EL    IMED
**                                              EREC  FIM                R.E
** OR
**
OR        D             D       BL
                 SREC  D        OUL   IMED      EA    LE
**                                              ERED  FIM                R.E
** EXCLUSIVE OR
**
EOR       D             D       BL
                 SREC  D        OUE   IMED      EA    LE
                                                ERED  FIM                R.E
```

BIBLIOGRAFIA

BIBLIOGRAFIA

1 CHU, Charles M., e OLDS, Carl L., "Microprogramming as a Technique to Minomize Card Types in High Performance Machine Design", 5th. Annual Workshop an Microprogramming, University of Illinois, Urbana, Illinois, Setembro, 1972 (Preprints), pgs. 72 - 75.

2 CHU, Yaohan, "Computer Organization and Microprogramming", Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1972.

3 FLYNN, M.J., e ROSIN, R.F., "Microprogramming: An Introduction and Viewpoint", IEEE Transactions on Computers, vol. C-20, Julho, 1971, pgs. 727 - 731.

4 HINSHAW, David e IRANI, Keki, "Optimal Selection of Functional Components for Microprogrammable Central Processing Units", 5th. Annual Workshop on Microprogramming,University of Illinois, Urbana, Illinois, Setembro, 1972, pgs 8 - 27.

5. HUSSON, Samir S., "Microprogramming: Principles and Practices", Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1970.

6 JONES, Louise H., "A Survey of Current Work in Microprogramming", IEEE Computer Society, Agosto, 1975, pgs. 33 - 37.

7 KLEIR, R. L., e RAMAMOORTHY, C.V., "Optimization Strategies for Microprograms", IEEE Transactions on Computers, vol. C-20, Nº 7, Julho, 1971, pgs. 783 - 794.

8 LESSER, V. R., "An Introduction to the Direct Emulation of Control Structures by a Parallel Microcomputer", IEEE Transactions on Computers, vol. C-20, Nº 7, Julho, 1971, pgs. 751 - 763.

9 MALLACH, Efren G., "Emulator Architecture", IEEE Computer Society, Agosto, 1975, pgs. 24 - 31.

10 MANDEL, R.L., "Hardware-Software Trade-Offs - Reasons and Directions", AFIPS Conference Proceedings, vol. 41, Part I, 1972, pgs. 453 - 459.

11 PAINKE, Helmut, "Initialization of Microprogrammed Machines", 5th. Annual Workshop on Microprogramming, University of Illinois, Urbana, Illinois, Setembro, 1972, (Preprints), pgs. 3 - 7.

12 REDFIELD, Stephen R.,"A Study in Microprogrammed Processors: A Medium Sized Microprogrammed Processor", IEEE Transactions on Computers, vol. C-20, Nº 7, Julho, 1971, pgs. 743 - 750.

13 ROSIN, R.F., FRIEDER, G., e ECKHOUSE Jr., R.H., "An Environment for Research in Microprogramming and Emulation", 4th Workshop on Microprogramming, Santa Cruz, Setembro, 1971, (Prepints), pgs. 342 - 381.

14 ROSIN, R.F., "Contemporary Concepts of Microprogramming and Emulation", Computer Surveys 1, Dezembro, 1969, pgs. 197 - 212.

15 SAAL, Harry J., e SHUSTEK, Leonard J., "Microprogrammed Implementation of Computer Measurement Techniques", 5th Annual Workshop on Microprogramming, University of Illinois, Urbana, Illinois, Setembro, 1972, (Preprints), pgs. 42 - 50.

16 SALISBURY, Allan B., "The Evaluation of Microprogram Implemented Emulators", Digital Systems Laboratory, Stanford University, California, 1973.

17 SCHOEN, Thomas A., e BELSOLE Jr., Michael R., "A Burroughs 220 Emulator for the IBM 360/25", IEEE Transactions on Computers, vol. C-20, Nº 7, Julho, 1971, pgs. 795 - 797.

18 TUCKER, S.G., "Emulation Techniques", 4th Workshop on Microprogramming, Santa Cruz, Setembro, 1971, (Preprints), pgs. 398 - 418.

19 WILKES, Maurice V., "The Use of a Writable Control Memory in a Multiprogramming Environment", 5th Annual Workshop on Microprogramming, University of Illinois, Urbana, Setembro, 1972, (Preprints), pgs. 62 - 65.

20 "HP 2100 Computer: Microprogramming Guide", Hewlett Packard Company, Cupertino, California, 1972.

21 "IBM 1130 Functional Characteristics", IBM System Reference Library, 1972.

22 "Manual de Arquitetura do G-10", FDTE, Escola Politécnica, USP, 1975.

23 "Manual de Microprogramação do G-10", FDTE, Escola Politécnica, USP, 1975.

24 "Manual de Operação do G-10", FDTE, Escola Politécnica, USP, 1975.

FD-104

FD

Autor: Grassiani, Edit

Título: Procedimentos Micorprogramados
num Minicomputador.

FD - 104

Grassiani, Edit
Procedimentos microprogramados
num minicomputador.

ESCOLA POLITÉCNICA - USP

BIBLIOTECA
Dept.º Eng. ELETRICIDADE
E. P. U. S. P.

VOLVER NA ÚLTIMA DATA MARCAI

FD-104

Grassiani, Edit
Procedimentos Microprogramados num Minicomputador.